



EPS

Escola Politècnica

Superior

Projecte/Treball Fi de Carrera

Estudi: Enginyeria Informàtica. Pla 1997

Títol: Algorismes per a la resolució de problemes de proximitat en terrenys

Document: Memòria

Alumne: Eduard Oliver Juanola

Director/Tutor: Joan Antoni Sellarès Chiva
Departament: Informàtica i Matemàtica Aplicada
Àrea: LSI

Convocatòria (mes/any): Febrer - 2007

Índex

1. Introducció i objectius	4
1.1 Treball previ	5
1.2 Objectius	6
1.3 Motivacions.....	8
1.4 Estructura de la memòria.....	8
1.5 Aplicacions.....	10
2. Estudi Previ.....	13
- Fonaments Teòrics.....	14
2.1 Introducció.	14
2.2 Terrenys.....	16
2.2.1 Introducció i definició	16
2.2.2 Triangulació de Delaunay	17
2.2.3 Estructura DCEL	19
2.3 Treballs anteriors	22
2.4 Preliminars	22
2.4.1 Algorisme de Dijkstra	22
2.4.2 Diagrames de Voronoi	24
2.5 Algorisme exacte de trobar camins mínims	25
2.5.1 Descripció de l'algorisme	25
2.5.2 Estructura de les finestres	26
2.5.3 Propagació i intersecció de finestres	27
2.5.4 Reconstrucció camí mínim (backtracing)	28
2.6 Terrenys amb més d'una seu	29
2.6.1 Representació Diagrames de Voronoi.....	29
2.6.2 Anàlisi de complexitat.....	31
- Fonaments Pràctics.....	33
2.7 Introducció	32
2.8 La llibreria Qt (KDE)	32
2.9 La llibreria OpenGL.....	33
2.10 La llibreria CGAL	35
2.11 KDevelop	36
2.12 QtDesigner	38
2.13 Objectteering UML Modeler.....	39
3. Requeriments del sistema.....	41
3.1 Requeriments funcionals	42
3.1.1 Actors del sistema.	46
3.1.2 Diagrames de casos d'ús	47
3.1.3 Fitxes de casos d'ús.	50
3.1.4 Diagrames d'activitat	53
3.2 Requeriments no funcionals	56
4. Anàlisi del sistema	59
4.1 Diagrama de classes	60
4.1.1 Paquet Interfície gràfica i programa principal.....	62
4.1.2 Paquet Terreny	65
4.1.3 Paquet Elements Visió i Distàncies.....	66

4.1.4	Paquet Events	67
4.1.5	Paquet Auxiliar	67
4.2	Diagrames de seqüència	68
4.2.1	Diagrama de seqüència Inserir seu al terreny	68
4.2.2	Diagrama de seqüència Camí òptim entre seu i punt	69
4.2.3	Diagrama de seqüència Diagrama de Voronoi sobre terreny	72
5.	Disseny del sistema.....	74
5.1	Disseny de la interfície	74
5.1.1	Editor 2D.....	76
5.1.2	Visor 3D	78
5.2	Format dels fitxers de terreny	80
5.3	Format dels fitxers d'imatge	81
6.	Implementació.....	82
6.1	Càlcul camins òptims	83
6.2	Diagrames de Voronoi	89
6.3	Jocs de proves	96
7.	Millores i Treball Futur	102
8.	Manual d'usuari	104
8.1	Instal·lació i execució del programari	105
8.2	Elements de la Interfície.....	106
8.2.1	Editor 2D.....	106
8.2.2	Visor 3D	110
8.3	Opcions del programa	114
8.3.1	Crear nou terreny.....	115
8.3.2	Crear nou terreny aleatori.....	116
8.3.3	Carregar un terreny.....	117
8.3.4	Desar un terreny	118
8.3.5	Guardar informació en format gràfic.....	119
8.3.6	Inserir seus en el terreny.....	120
8.3.7	Inserir elements restringits en el terreny.....	121
8.3.8	Inserir punt camí mínim en el terreny	124
8.3.9	Obtenir informació	125
8.3.10	Modificar el terreny.....	126
8.3.10.1	Modificar alçada de punts del terreny	126
8.3.10.2	Eliminar elements.....	126
8.3.11	Modificar configuració Editor 2D.....	127
8.3.12	Modes de treball (Visibilitat / Distàncies).....	128
8.3.13	Propagar terreny	129
8.3.14	Trobar camí més curt.....	130
8.3.15	Visualitzar elements visibles	131
8.3.16	Visualitzar Diagrames de Voronoi	132
8.3.17	Ajuda.....	133
8.3.18	Sortir de l'aplicació.....	134
8.3.19	Mostrar / Amagar Visor 3D.....	134
8.3.19.1	Opcions de navegació: moviment de la càmera	135
8.3.19.2	Realitzar zoom.....	136
8.3.19.3	Opcions de visualització.....	137
8.3.19.4	Guardar informació en format gràfic.....	137
8.3.19.5	Modificar configuració del Visor 3D	137
8.3.19.6	Veure finestres.....	138
8.3.19.7	Visualitzar elements visibles	139
8.3.19.8	Ajuda	140

8.3.19.9 Tancar el Visor 3D	141
Conclusions	142
Agraïments	146
Bibliografia	147
Adreces d'Internet	148
Índex de Figures	149
Annexos	153
Annex A Metodologia basada en UML	153
A.1 Etapes i activitats en el desenvolupament OO sobre UML.....	154
A.1.1 Descripció de requeriments i anàlisi del sistema.....	154
A.1.2 Disseny del sistema	156
A.1.3 Disseny detallat	156
A.1.4 Implementació i proves	157
A.2 Procés de desenvolupament de software OO	158
Annex B Nomenclatura UML	159
B.1 Nomenclatura utilitzada en els diagrames de casos d'ús	160
B.2 Nomenclatura utilitzada en els diagrames d'activitat	163
B.3 Nomenclatura utilitzada en els diagrames de classes	165
B.4 Nomenclatura utilitzada en els diagrames de seqüència	169

Capítol 1

Introducció i objectius

El modelatge, visualització i anàlisi de terrenys (visibilitat, càlcul de camins mínims, visualització diagrames de Voronoi...) és de gran importància en els Sistemes d'Informació Geogràfica (GIS). Actualment és de gran interès per a aquesta comunitat disposar de software que permeti analitzar terrenys. Aquest projecte final de carrera consisteix, a grans trets, en realitzar l'anàlisi, disseny i implementació d'un programari que sigui capaç de resoldre problemes de proximitat en terrenys, com poden ésser: determinació de camins òptims entre una o un conjunt de seus (que geomètricament es representen per punts, segments, polígons, poligonals, ...) i un o diversos punts, càlcul de la distància geodèsica entre una seu i un punt, visualització de Diagrames de Voronoi per a un conjunt de seus, Una part important per tal de poder obtenir tot això també ha d'ésser la de poder generar, visualitzar i modificar un model 3D d'un terreny a partir de dades introduïdes per l'usuari o obtingudes des d'un fitxer.

Per tal de poder construir l'aplicació desitjada ha calgut dissenyar una interfície gràfica d'usuari que permetés realitzar de forma interactiva la introducció, modificació i esborrat de les diferents seus (punts, segments, polígons, poligonals, ...) o restriccions del terreny, així com la seva visualització. El disseny de la interfície s'ha dut a terme tenint en compte una entrada de dades fàcil i intuïtiva, per tal que qualsevol tipus d'usuari pugui ésser capaç d'utilitzar-la. A partir d'aquesta interfície es podran dur a terme tots els objectius que ens havíem marcat en iniciar el projecte, i que principalment fan referència a la resolució de diversos problemes de proximitat en terrenys, així com a la seva corresponent visualització.

L'aplicació a modelar utilitzarà el paradigma de la programació d'orientació a objectes ja que el que interessarà en tot moment serà el comportament de cadascun dels objectes creats. Per aquest motiu les etapes d'anàlisi i de disseny de l'aplicació es realitzaran en aquest tipus de paradigma utilitzant la nomenclatura **UML** (veure Annex "*Nomenclatura UML*").

Cal a dir que, pel que fa a la interfície gràfica d'usuari s'ha partit de la interfície realitzada en el Projecte Final de Carrera "***Disseny d'una eina de modelat de terrenys i càlcul de corbes de nivell***", la qual es va ampliar i millorar en el Projecte Final de Carrera "***Algorismes de generació i modificació de terrenys***". Aquesta ha estat adaptada i modificada segons les necessitats del nostre projecte. En el *capítol 5 "Disseny del Sistema"* aprofundirem més en la interfície i veurem quins canvis s'hi han introduït per tal d'adaptar-la al nostre projecte.

En aquest primer capítol de la memòria donarem a conèixer la finalitat d'aquest projecte, és a dir, descriurem els principals objectius que es volen arribar a assolir. A més a més, s'esmentaran les motivacions referents al projecte i l'estructura que tindrà la memòria. Finalment, veurem algunes de les aplicacions que existeixen dins l'àrea específica de treball.

1.1 Treball previ

Tal i com ja hem comentat, abans de començar a entrar en el propi projecte, ens calia realitzar un estudi sobre el treball previ que s'havia realitzat amb la interfície del programa, així com conèixer les funcionalitats i opcions que aquesta ens oferia, per tal de veure quina era la part que es podia utilitzar, quina era la part que es podia modificar i quina era la part que s'havia d'afegir per tal d'adequar la interfície gràfica a la problemàtica del nostre projecte.

Observant el treball previ del qual es va partir, varem veure que aquest treball ens permetia principalment generar, visualitzar, editar i emmagatzemar terrenys reals o ficticis a través d'una interfície gràfica d'usuari amigable i senzilla d'utilitzar. Tot seguit descriurem breument cadascuna de les funcions més importants que ens oferia:

- **Generar un Terreny** : L'aplicació ens permet generar terrenys a partir d'unes dades inicials. Aquestes dades inicials hauran de poder ésser obtingudes, ja sigui de forma aleatòria, creades a partir d'un determinat algorisme, o bé, carregades des d'un fitxer. Com que es vol obtenir una triangulació que compleixi les propietats de ***Delaunay***, s'utilitza l'algorisme *Delaunay Conforming* per tal de poder triangular un *PSLG*.

- **Visualitzar el Terreny**: L'aplicació ens permet visualitzar el terreny tant en 2D com en 3D. Quan parlem de visualitzar un terreny en 2D ens referim a visualitzar el domini del terreny o la projecció al pla XY del terreny (que per defecte es 2'5D). La interacció amb el terreny per tal d'introduir-hi modificacions

es durà a terme a través de l'editor 2D. D'altra banda, la visualització del terreny en 3D aporta realisme a la imatge i dóna una idea sobre el resultat final de la interacció.

- **Editar el terreny:** L'edició de terrenys és una altra de les funcionalitats que ens ofereix la interfície. A través d'aquesta, podem introduir elements restringits per tal de modificar l'estructura de la malla. Aquests elements restringits sempre faran referència a nous vèrtexs o punts de la triangulació. Les restriccions que contindrà el *PSLG* seran sempre de tipus geomètric. Així doncs, els tipus que caldrà que l'aplicació tracti seran: punts, segments, polígons i poligonals. Les diferents operacions que ens ofereix l'edició de terrenys són:

- Introduir / Eliminar / Moure elements restringits (punts, segments, polígons, poligonals) dins del terreny.
- Aplicar color a les cares del terreny

- **Emmagatzemar el terreny:** L'aplicació ens permet poder emmagatzemar el terreny creat o editat en fitxers. Els formats que l'aplicació suporta són :

- Formats estàndards
- Formats propis de l'aplicació

1.2 Objectius

L'objectiu principal del projecte és desenvolupar una aplicació per a la **resolució de diversos problemes de proximitat en terrenys**. Per tal d'arribar a aquest objectiu final, ens caldrà bàsicament:

- Construir una interfície que ens permeti inserir i escollir seus (punts) sobre un terreny.
- Dissenyar i implementar diferents algorismes per tal de determinar camins òptims sobre terrenys.
- Calcular la distància geodèsica entre un punt del terreny i la seva seu més propera.
- Poder visualitzar tant en 2D com en 3D camins òptims sobre terrenys.
- Calcular i visualitzar Diagrames de Voronoi sobre terrenys.

Així doncs, podríem dir que els objectius més importants i amb els que s'ha hagut de treballar de forma exhaustiva a l'hora de realitzar aquest projecte són aquests que acabem d'anomenar anteriorment. Per tant, per poder arribar al nostre gran objectiu final, haurem d'assolir aquests quatre objectius anteriors.

No obstant això, per arribar fins a aquests objectius, primerament ens calia disposar d'un programari que ens permetés generar, visualitzar, editar i emmagatzemar terrenys reals o ficticis a través d'una interfície gràfica d'usuari amigable i senzilla d'utilitzar. Tal com acabem de veure en l'apartat anterior, això ja ho tenim al nostre abast, tot i que algunes de les funcionalitats que se'ns ofereixen inicialment s'hauran de refinar o ampliar:

- Visualitzar el Terreny: A més de poder visualitzar el terreny generat, l'aplicació ens ha de permetre visualitzar les seues que s'insereixin, s'esborrin o es modifiquin del terreny, així com tots els canvis que introduïm en el terreny.

- Editar el Terreny: Per tal de poder resoldre els nostres problemes de proximitat, ens caldrà poder editar el terreny. Així doncs, també hem de poder introduir i eliminar seues en el terreny. Les seues seran de tipus geomètric, i poden ésser: punts, segments, polígons i poligonals, tot i que l'objectiu inicial d'aquest projecte només inclou el treball amb punts. També hem de poder inserir i eliminar punts que no actuïn com a seu ni com a vèrtexs restringits de la triangulació, punts que anomenarem de camí mínim.

- Emmagatzemar el Terreny: L'aplicació ens ha de permetre poder emmagatzemar el terreny creat o editat en fitxers per tal que la informació i els càlculs de camins òptims o Diagrames de Voronoi que haguem realitzat puguin ésser recuperats posteriorment quan ens interessi.

Apart d'aquests objectius, el programari també ens ha de permetre obtenir informació sobre la triangulació, com per exemple poden ésser les coordenades d'un vèrtex. Com es que es tracta d'una interfície gràfica d'usuari, caldrà que aquesta disposi de certs paràmetres que l'usuari pugui configurar, fent que cada usuari pugui personalitzar-se mínimament la interfície al seu gust. També s'han de poder obtenir captures d'imatges del terreny (tant en 2D com en 3D) en diferents formats gràfics.

Un altre objectiu important és el fet d'aprendre i utilitzar llibreries de domini públic. Bàsicament he utilitzat les del llenguatge de programació **C++**, **OpenGL** (sistema que permet crear qualsevol tipus de gràfics), **Qt** (multiplataforma que permet crear interfícies gràfiques d'usuaris (GUI) utilitzant el llenguatge de programació C++), **CGAL** (llibreries de domini públic desenvolupades per diversos grups que treballen el Geometria Computacional), etc. Pel que fa als programes utilitzats, s'ha treballat principalment amb les eines **KDevelop**, **QtDesigner** i el **Objecteering UML Modeler** per tal de realitzar tota l'estructura de classes i els corresponents diagrames UML. Posteriorment, en l'apartat *fonaments pràctics* del *Capítol 2* s'explicaran amb més profunditat totes les llibreries i el software utilitzats.

1.3 Motivacions

Una de les principals motivacions per les quals vaig escollir aquest projecte va ésser el fet de tenir la possibilitat de conèixer nous àmbits en el món de la informàtica i poder treballar en la vessant de la *Geometria Computacional*, gairebé desconeguda per a mi fins aleshores.

També vaig escollir aquest projecte perquè se m'oferia la possibilitat de formar part d'un equip de treball amb gent experta en la matèria que ja fa temps que està treballant en aquesta vessant, la qual cosa feia que em pogués sentir recolzat en tot moment i tingués bastant de suport quan em sorgissin problemes i/o inconvenients.

Una altra de les motivacions ha estat el fet de poder obtenir o ampliar coneixements sobre algunes de les llibreries de codi lliure que s'hauran d'utilitzar per poder realitzar el projecte, com poden ésser les *Qt*, les *OpenGL* o les *CGAL*. Finalment dir també que aquest projecte em servirà com a experiència com a programador i dissenyador, especialment en la part de saber resoldre qualsevol tipus de problema que pugui sorgir durant la realització d'aquest projecte i per tal de disposar de nous coneixements o ampliar coneixements ja existents.

1.4 Estructura de la memòria

La memòria ha estat estructurada en els següents capítols:

Capítol 1. Introducció i objectius

Introducció dels primers conceptes bàsics dels Sistemes d'Informació Geogràfics (GIS) i d'algunes de les aplicacions dins l'àrea específica de treball. Descripció dels objectius i les motivacions del projecte.

Capítol 2. Estudi Previ

Presentació dels coneixements que s'han hagut d'adquirir per tal d'assolir els objectius del projecte. Explicació dels diferents aspectes teòrics que engloben el projecte, així com els pràctics, que fan referència a les eines i llibreries que s'han usat per desenvolupar el projecte.

Capítol 3. Requeriments del Sistema

En aquest capítol realitzarem l'anàlisi dels requeriments que hauria de complir el sistema, tant funcionals com no funcionals. Respecte als requeriments funcionals de l'aplicació, s'utilitzaran diverses eines de modelat UML, com poden ésser els diagrames de casos d'ús, així com les

seves corresponents fitxes. També es realitzarà algun diagrama d'activitat per tal de complementar i/o substituir les fitxes dels casos d'ús. Així, a través d'aquest capítol sabrem què ens caldrà tant des del punt de vista de software com de hardware per tal que el nostre projecte pugui complir amb els seus objectius i pugui ésser executat i utilitzat sense problemes.

Capítol 4. Anàlisi del Sistema

En aquest capítol explicarem l'estructura de classes dissenyada per implementar el projecte. Ho reforçarem amb diferents diagrames UML que ens permetran il·lustrar el comportament del sistema. Veurem principalment el diagrama de classes del sistema, juntament amb diagrames de seqüència de les funcionalitats més complexes.

Capítol 5. Disseny del Sistema

En aquest capítol veurem principalment quin ha estat el disseny de l'aplicació i els canvis efectuats a la interfície gràfica inicial per tal de poder resoldre els objectius del projecte.

Capítol 6. Implementació

En aquest capítol explicarem com ho hem fet per tal d'implementar els diferents problemes de proximitat en terrenys que teníem (determinar camins òptims entre seu i punt, visualitzar els Diagrames de Voronoi per a un conjunt de seus, ...) Per a cada problema, veurem alguns exemples d'imatges visuals generades amb el mètode corresponent.

Capítol 7. Millores i Treball Futur

En aquest capítol comentarem les possibles millores que es podrien fer sobre els problemes de proximitat resolts, així com també comentarem el treball futur que es podria fer per tal de millorar les prestacions de la interfície.

Capítol 8. Manual d'usuari

En aquest capítol explicarem el manual d'usuari. A través d'aquest manual s'especificarà el funcionament de cadascun dels elements de l'aplicació (tant menús com botons). D'aquesta manera, qualsevol usuari podrà utilitzar el programa simplement consultant aquest manual. A més a més, es troba acompanyat d'imatges que fan que sigui molt més entenedor i pràctic.

Conclusions

En aquest apartat repassarem els objectius que ens havíem marcat en l'inici del projecte i mirarem si els hem assolit amb èxit. Traurem també les conclusions més significatives a les que haurem arribat després de desenvolupar tot el projecte.

Agraïments

Disposarem d'un espai per tal de realitzar els agraïments corresponents a les persones que m'han ajudat a realitzar el projecte i sense les quals tot això no hagués estat possible.

Bibliografia

En aquest apartat veurem quina ha estat la bibliografia consultada.

Adreces d'Internet

En aquest apartat veurem quines han estat les adreces d'Internet consultades per tal de dur a bon terme el projecte.

Índex de Figures

Índex de les pàgines on podem trobar les figures que hi ha al llarg dels diferents capítols de la memòria.

Annexos

Disposarem de dos annexos per tal d'ampliar algun dels conceptes utilitzats al llarg de la memòria en diferents capítols. Hi trobarem la descripció de : *Metodologia basada en UML (Annex A)* i *Nomenclatura UML (Annex B)*.

1.5 Aplicacions

Cada dia disposem de més informació al nostre abast i part d'aquesta informació millora la seva utilitat si li donem una visió geogràfica. Tenint en compte aquest fet, apareix **GIS** (*Sistemes d'Informació Geogràfica*) el qual ens permet mostrar la informació des d'un punt de vista geogràfic. Els sistemes GIS són sistemes complexos basats en bases de dades i un software de representació gràfica i tractament de dades espacials, com per exemple *Oracle* amb *Spatial option*.

El GIS disposa d'un gran nombre d'aplicacions, entre les quals podem trobar-hi: agricultura, gestió dels recursos naturals, cadastre, planificació i gestió de serveis públics, aplicacions urbanes, aplicacions cartogràfiques, defensa i seguretat, cens i estadístiques de població, anàlisi de mercat, ...

En la segona meitat dels anys 70 l'estudi dels algorismes geomètrics va donar lloc al naixement d'una disciplina coneguda com **Geometria Computacional**. La motivació per l'estudi dels algorismes geomètrics prové d'àrees com la robòtica, la computació gràfica, la manufacturació automatitzada i el GIS.

Els GIS inclouen molts problemes relacionats amb la geometria tal com l'emmagatzematge, recuperació i visualització de dades espacials, superposició de mapes, interpolació espacial i generalització.

Existeixen diverses estructures i algorismes de geometria estàndards que amb o sense modificacions poden ser utilitzats en GIS. A més a més, la quantitat de dades és tan gran que calen algorismes eficients.

Tot seguit es mostraran una sèrie d'aplicacions de GIS fent referència a cadascuna de les àrees de la geometria que els hi aporten solució:

- Suposem que volem desenvolupar un navegador de cotxes. Això requereix emmagatzemar un mapa molt gros de rutes i altres dades. En qualsevol moment hem de poder determinar la posició del cotxe sobre el mapa i ràpidament seleccionar-ne una petita porció per mostrar-la sobre l'ordinador d'abord. Els algorismes necessaris serien: localització d'un punt, cares de la triangulació contingudes dins d'un rectangle, ...



Figura 1.1 : Navegador de cotxe

- La informació relacionada amb l'alçada d'un terreny muntanyós es troba només disponible en certs punts del terreny. Per altres posicions hem d'obtenir les alçades interpolant els punts propers de la mostra. En aquest cas utilitzarem la triangulació de **Delaunay**.
- En el cas que vulguem localitzar el telèfon públic més proper, per exemple, requerirem del càlcul d'un **diagrama de Voronoi**.
- La combinació de diferents tipus de dades és una de les operacions més importants en GIS. Per exemple, ens podem preguntar quines cases es troben dins d'un bosc, localitzar tots els ponts situats on creuen rius i carreteres o determinar una bona localització per un camp de golf, trobant una àrea lleugerament elevada, barata i no massa lluny de la ciutat. Per combinar les dades cal superposar diferents mapes i utilitzar la intersecció entre segments.

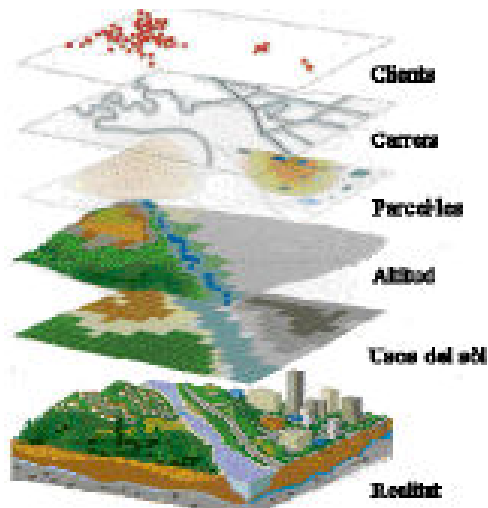


Figura 1.2 : Combinació de dades per capes

- La modificació de terrenys és una part important per moltes aplicacions com el modelat de paisatges, l'enginyeria civil, ... En el món real les modificacions del paisatge es duen a terme “*esculpint*” el sòl a través d'un equipament pesat. Abans de començar a treballar sobre el terreny real, moltes vegades és necessari simular i avaluar primer el procés. En el disseny digital l'enginyer ha de poder visualitzar els efectes de les modificacions proposades sobre el terreny podent-les, d'aquesta manera, ajustar. Els principals aspectes que cal tenir en compte són els efectes visuals, com per exemple en l'arquitectura de paisatges, la forma de la superfície desitjada i el volum mogut, com en el cas de la construcció de carreteres, i el resultat dels canvis en les pendents, la potencial erosió, el modelat hidrogràfic per la construcció de pantans, ... L'habilitat per modelar interactivament paisatges, visualitzar els efectes i posteriorment ajustar el model, té un gran valor en diversos camps de treball. És en aquesta àrea de recerca en la qual podríem situar el nostre projecte, en especial en relació a la modificació i visualització interactiva.

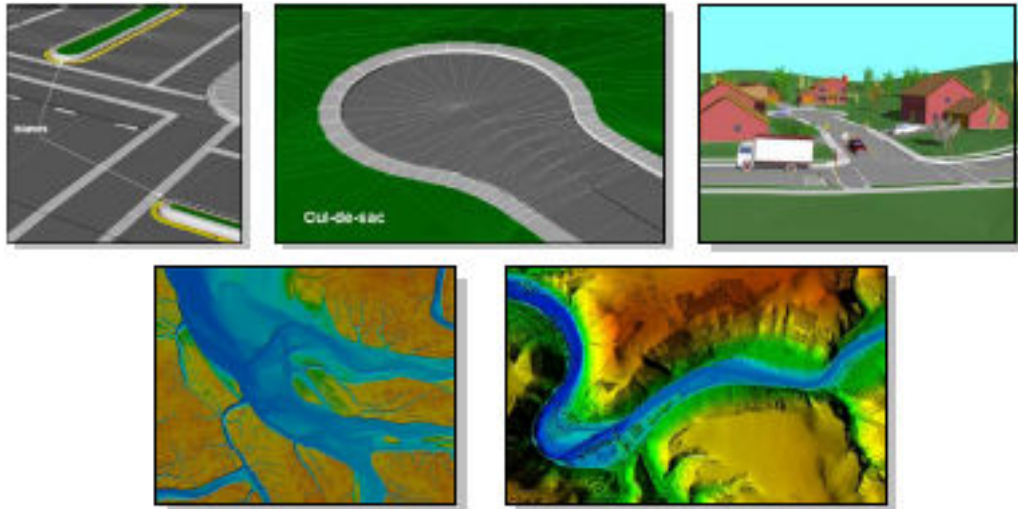


Figura 1.3 : Exemples d'aplicacions

Capítol 2

Estudi Previ

En aquest apartat de la memòria presentaré l'estudi previ que he hagut de realitzar per tal de desenvolupar aquest projecte. Molta de la problemàtica i el context que envoltava el problema era gairebé desconegut per a mi fins al moment, i també una part de les eines amb les quals he hagut de treballar resultaven gairebé noves per a mi en el moment de començar el treball. Per tant, una part de la feina en el procés de realització del projecte ha estat també l'adquisició de tots aquests coneixements.

El capítol ha estat estructurat en dos blocs diferents. En un primer presentarem els fonaments teòrics relacionats amb els problemes de proximitat en terrenys. Primerament explicarem què són els terrenys i com els podem representar, així com també veurem quines maneres hi ha per tal d'obtenir camins òptims sobre terrenys i calcular-ne la distància mínima. Finalment veurem què són els diagrames de Voronoi i com ho podem fer per tal de calcular-los i poder-los visualitzar en un terreny.

En la segona part del capítol es presentaran de forma sintetitzada les eines i llibreries que s'han utilitzat per tal de desenvolupar tota la implementació. Ens centrarem bàsicament en el funcionament de les llibreries **Qt**, les **OpenGL**, les **CGAL** i tot el **software** que s'ha necessitat per tal de desenvolupar el projecte.

Fonaments Teòrics

2.1 Introducció

El fet de trobar i determinar camins òptims en superfícies polièdriques és un dels problemes fonamentals de la geometria computacional, tenint actualment diverses aplicacions en informàtica gràfica, robòtica i sistemes d'informació geogràfica.

Si considerem P com a una superfície polièdrica (incloent la possibilitat que aquesta sigui no-convexa) formada per n cares triangulars, el problema de trobar el camí més curt respecte una seu de la superfície polièdrica, consisteix en trobar el camí més curt, en termes de distància Euclidiana, d'una seu a qualsevol altre punt objectiu del terreny tal que el seu camí es quedi dins de la superfície polièdrica P .

En la referència [MMP87] Mitchell i companyia varen presentar un algorisme per tal de solucionar el problema del camí més curt d'una seu desenvolupant un mètode de Dijkstra continu, que propagava les distàncies des de la seu fins a la resta de la superfície polièdrica P . L'algorisme construeix en un temps de l'ordre $O(n^2 \log n)$ una estructura de dades que codifica implícitament els camins més curts des d'una seu donada a tota la resta de punts de P .

En la referència [SSKGH05] es descriu una manera simple de implementar l'algorisme de trobar els camins mínims, així com es mostra que normalment s'executa més ràpid en les superfícies polièdriques que el temps que en el pitjor dels seus casos indica, que és $O(n^2 \log n)$ (Veure *Figura 2.1* per saber amb quins exemples es varen realitzar les proves). Posteriorment, *Chen i Han* (veure referència [CH96]), utilitzant un aproximament una mica diferent, van millorar-lo fins a aconseguir un temps de l'algorisme de l'ordre de $O(n^2)$. Més endavant, els investigadors *Kaneva i O'Rourke* (veure referència [KO00]) van implementar l'algorisme ideat per *Chen i Han* i van comunicar que la implementació era difícil per a superfícies polièdriques no convexes, i que la mida de memòria era un factor que podia limitar l'algorisme. Finalment, *Kapoor* [Kapoor93] va presentar un algorisme seguint el paradigma del mètode de Dijkstra continu que calculava el camí més curt d'una seu a un punt objectiu en temps $O(n \log^2 n)$. Això però, s'aconseguia amb un algorisme molt complicat i que mai ha estat clar si va arribar a ésser implementat.

En aquest projecte, presentarem un algorisme per tal de calcular els camins mínims exactes, i en conseqüència també les distàncies exactes, des d'una seu d'un terreny, que es pot veure com a un cas particular d'una superfície polièdrica, representada com a una malla triangular. L'algorisme fàcilment es podrà estendre al cas d'un conjunt de seus, fet que ens permeti realitzar posteriorment una representació del *Diagrama de Voronoi* d'un conjunt de punts (seus) del terreny, cosa que ens permetrà obtenir el camí més curt des de qualsevol punt del terreny a la seva seu més propera. Així doncs, hem implementat l'algorisme basant-nos en les directrius que s'expliquen en la referència [SSKGH05] sobre l'algorisme d'ordre $O(n^2 \log n)$ proposat per Mitchell [MMP87]. En la referència [SSKGH05] podem observar com està demostrat que l'algorisme, a la pràctica, sol treballar en temps sub-quadràtic. Per exemple, es pot calcular la distància geodèsica exacta d'una seu a la resta de vèrtexs d'un triangulació de 400K-triangles en aproximadament 1 minut.



Figura 2.1 : Exemples utilitzats per calcular temps d'execució en la referència [SSKGH05]

En el nostre cas, tal i com ja hem comentat, treballarem sempre amb **terrenys**, ja que uns dels nostres principals objectius que voldrem resoldre seran els de determinar camins òptims en terrenys i visualitzar diagrames de Voronoi en el terreny. Destaquem el fet que podem veure un terreny com a un cas particular de superfície polièdrica. Així doncs, abans d'entrar en més detalls sobre l'estructura de l'algorisme utilitzat per trobar camins mínims, anem a descriure què és un terreny, com el podem representar i quina és l'estructura de dades que ens permet emmagatzemar un terreny.

2.2 Terrenys

2.2.1 Introducció i definició

Podríem modelar qualsevol part de la superfície de la terra com a un **terreny**. Segons la referència [BKOO97], podríem definir un terreny com una superfície de dos dimensions en un espai tridimensional amb una propietat especial :

Tota línia vertical interseca la superfície en un punt

Una definició més formal de **terreny** és la següent:

Un terreny és un graf d'una funció $f : A \subset \mathbb{R}^2 \rightarrow \mathbb{R}$ que assigna una alçada $f(p)$ per a tot punt p en el domini A d'un terreny

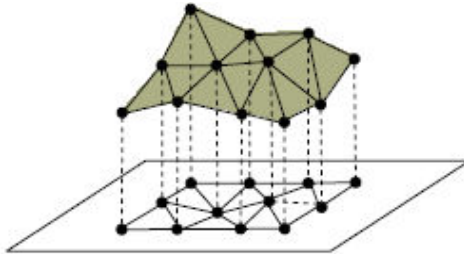


Figura 2.2 : Esquema d'un terreny

Les mesures de l'alçada d'un terreny real són realitzades a través d'un mostreig. D'aquesta manera, el nostre coneixement sobre f es troba restringit a un conjunt finit $P \subset A$ corresponent als punts d'una mostra. Per tal d'aproximar les dades desconegudes, es proposa una triangulació de P . Després de triangular els punts de P en dos dimensions, cada vèrtex de la triangulació és pujat a l'alçada corresponent, fent que tot triangle es trobi dins de l'espai tridimensional. Existeixen diferents maneres de triangular aquest conjunt P . Entre aquestes triangulacions, la **triangulació de Delaunay** assegura que els punts més propers entre ells pertanyeran al mateix triangle.

Els terrenys no són considerats superfícies tridimensionals ja que es troben definits per una funció, tal i com hem vist. Per aquest motiu, cada punt del domini A tan sols pot tenir una alçada, fet que no es presenta a l'espai, on un punt pot tenir més d'un valor. Per tant, direm que un **terreny** és un model 2.5D que no pot representar xarxes no planars o estructures complexes, com per exemple túnels, ponts o edificis.

2.2.2 Triangulació de Delaunay

En dos dimensions, una triangulació d'un conjunt V de vèrtexs és un conjunt T de triangles on la unió dels seus vèrtex formen V , l'interior dels quals no intersecciona entre ells.

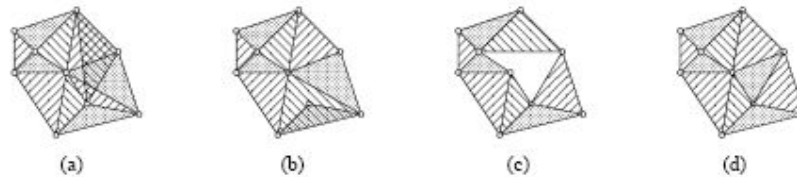


Figura 2.3 : En els casos (a) i (b) els triangles interseccionen, en (c) existeix un polígon simple que no és un triangle i (d) és una triangulació correcta.

La **Triangulació de Delaunay** D de V va ésser definida per *Boris N. Delaunay* l'any 1934 com un graf en el qual qualsevol cercle en el pla s'anomenarà *buit* si aquest no conté cap vèrtex de V en el seu interior (no es consideren interiors els vèrtexs que es trobin sobre el cercle). Suposem que u i v són un parell de vèrtexs, llavors l'aresta uv es troba dins de D si i només si existeix com a mínim un cercle buit que passa per sobre de u i v . Tota aresta que satisfaci aquesta propietat és anomenada **Delaunay** (referència [Shew97])

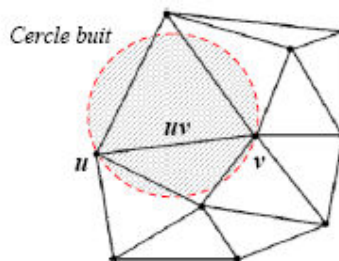


Figura 2.4 : Definició d'aresta Delaunay

Tota aresta de l'envolupant convexa o frontera d'un conjunt de vèrtexs és *Delaunay*, tal i com es pot observar en la següent figura. Per a qualsevol aresta e de l'envolupant convexa és sempre possible trobar un cercle buit que contingui e , començant amb el cercle més petit de e i fent-lo créixer fora de la triangulació.

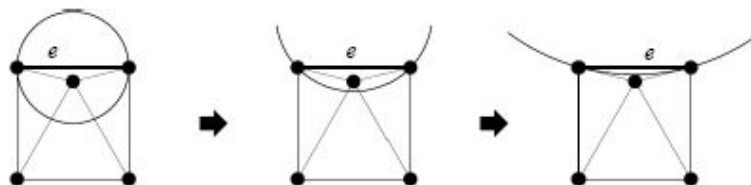


Figura 2.5 : Cadascuna de les arestes que formen l'envolupant convexa és Delaunay, ja que sempre és possible trobar un cercle buit que passi a través dels seus extrems.

Si un dels vèrtexs de V es troba en *posició general*, (Figura 2.6) és a dir, que no es troba sobre la mateixa circumferència definida per uns altres tres vèrtexs, la *triangulació de Delaunay* és única. Com a primer pas per a garantir la unicitat de la triangulació de *Delaunay* donarem la noció de **triangle Delaunay**. El circumcercle d'un triangle és l'únic cercle que passa per sobre dels seus tres vèrtexs. Un triangle serà anomenat *Delaunay* si i només si el seu circumcercle és buit. Aquesta característica, il·lustrada en la Figura 2.7 és anomenada *propietat del circumcercle buit o propietat Delaunay*.

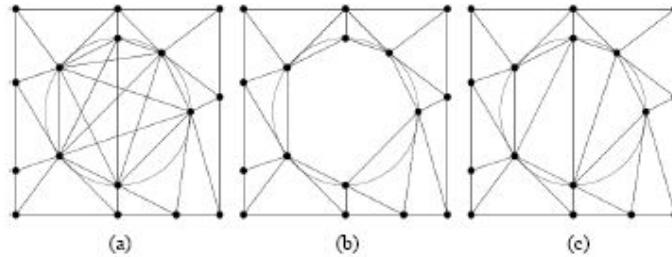


Figura 2.6 : Com que els punts no es troben en posició general ja que existeix una circumferència que passa per més de tres punts (b) existeixen diverses triangulacions Delaunay possibles, tal i com es mostra en la figura (a). En la figura (c) s'escull un subconjunt d'arestes Delaunay per formar una triangulació

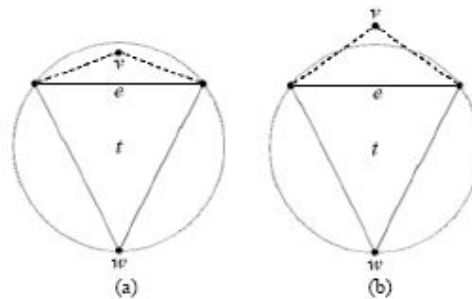


Figura 2.7 : Si el triangle t no és Delaunay (a) almenys un dels vèrtexs de V està contingut pel circumcentre de t . En la figura (b) t és Delaunay

La **triangulació de Delaunay**, tal i com es pot observar en la següent figura, és una estructura geomètrica d'entre totes les triangulacions existents d'un conjunt de punts que: *maximitza l'angle mínim en la triangulació, redueix al mínim el major circumcercle i minimitza el cercle de contenció mínim més gran*.

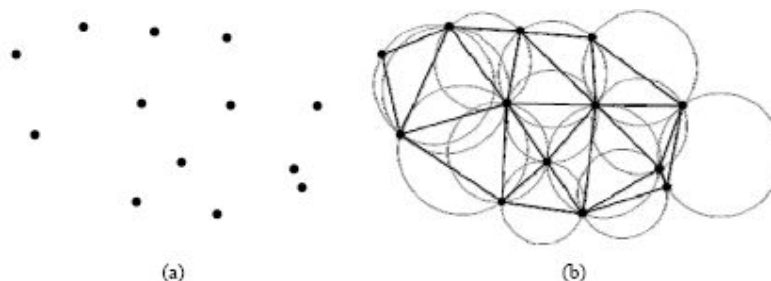


Figura 2.8 : A partir d'un núvol de punts (a) obtenim una triangulació de Delaunay (b) on tot triangle té un circumcentre buit

Cal afegir també que la **triangulació de Delaunay** és el dual del **diagrama de Voronoi**, és a dir, a partir d'una *triangulació de Delaunay* podem obtenir el corresponent *diagrama de Voronoi* i viceversa, tal i com es pot observar en la *Figura 2.9*. Un **diagrama de Voronoi**, també anomenat *diagrama de proximitat*, d'un conjunt de vèrtexs és una subdivisió del pla en regions *poligonals*, algunes de les quals són infinites, on cadascuna de les regions és un conjunt de punts en el pla que es troben més propers a un vèrtex d'entrada que a qualsevol altre.

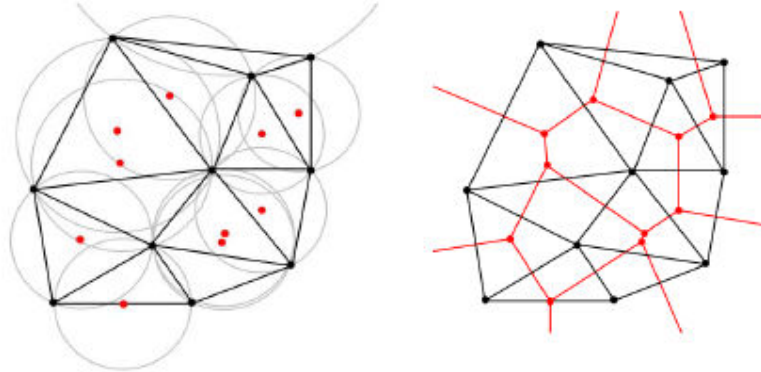


Figura 2.9 : Triangulació de Delaunay d'un núvol de punts amb els circumcentres buits i en vermell els circumcentres de cadascun d'ells que connectats formaran el Diagrama de Voronoi (b) també en vermell

Existeixen diferents tipus d'algorismes per tal de poder construir **triangulacions de Delaunay** en 2 dimensions. Alguns d'aquests podrien ésser: divideix i venç, escombrat, incremental, ... Tots ells disposen d'un temps de càlcul logarítmic de l'ordre $O(n \log n)$. No obstant això, utilitzant el mètode optimitzat del divideix i venç podem obtenir un temps lineal $O(n)$ en el cas que disposem de punts uniformement espaiats.

2.2.3 L'estructura DCEL

Per tal d'emmagatzemar la informació de la triangulació, utilitzarem l'estructura de dades **DCEL** (*Doubly Connected Edges List*), (veure referències [BKOO97] i [PrepSha85]), que ens permetrà guardar la informació sobre les relacions topològiques existents entre **vèrtexs**, **cares** i **arestes**. En l'estructura **DCEL**, les arestes que es troben en la subdivisió són representades a través de dos elements coneguts com arestes orientades, de manera que mentre una aresta orientada envolta una de les cares incidents, l'altre permet recórrer l'altre cara adjacent. Suposant que e fos una aresta orientada, llavors el $Twin(e)$ fa referència a l'altra aresta orientada oposada que forma part de la mateixa aresta. En resum, les arestes que envolten una cara es poden recórrer circularment en un sentit o altre. En la següent figura podem observar les característiques anteriorment descrites.

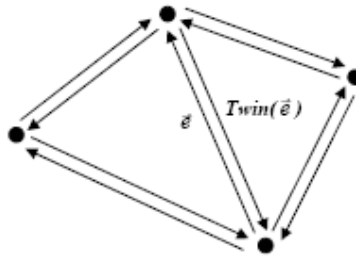


Figura 2.10 : Esquema d'una subdivisió planar on e i $Twin(e)$ són les arestes orientades oposades d'una mateixa aresta

L'estructura *DCEL* pot ésser definida a través de tres taules que permeten enregistrar les característiques que defineixen les arestes orientades, els vèrtexs i les cares.

Vèrtexs:

Per a cada **vèrtex** de la subdivisió planar s'emmagatzemen les seves coordenades, juntament amb un punter a qualsevol de les arestes orientades que tingui el vèrtex com a origen. Aquest últim serà anomenat *Incident Edge*.

Arestes Orientades:

Cada **aresta** de la subdivisió planar es representa per dues arestes orientades, que permeten definir la cara que es troba a la dreta o a l'esquerra de l'aresta. Cada aresta orientada té un punter a la seva aresta orientada oposada, anomenada *Twin*. A més a més, s'emmagatzema un punter al vèrtex *Origin*. També s'emmagatzema un punter a la cara situada a l'esquerra de l'aresta, anomenada *Incident Face* (cara incident). Finalment, es guarden les arestes orientades següent *Next* i prèvia *Prev* de la cara incident, al voltant de la cara en sentit negatiu (contrari al moviment de les agulles del rellotge). Els punters *Next* estan ordenats al voltant de la cara en el sentit negatiu i els punters *Prev* en sentit positiu. A partir d'aquests podrem recórrer totes les arestes d'una cara. En la *Figura 2.11* podem observar les característiques anteriorment descrites:

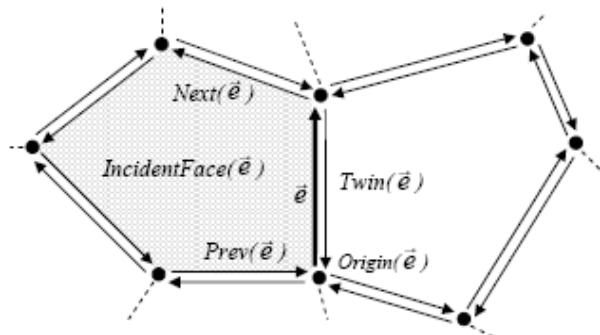


Figura 2.11 : Informació emmagatzemada al DCEL per cada aresta orientada

Cares:

Cadascuna de les **cares** de la subdivisió planar emmagatzema únicament un punter a una aresta orientada. Cal que el punter a la cara actual sigui equivalent al que tingui l'aresta orientada com a *Incident Face*.

Per acabar d'entendre del tot l'estructura de dades **DCEL** podem observar la figura següent.

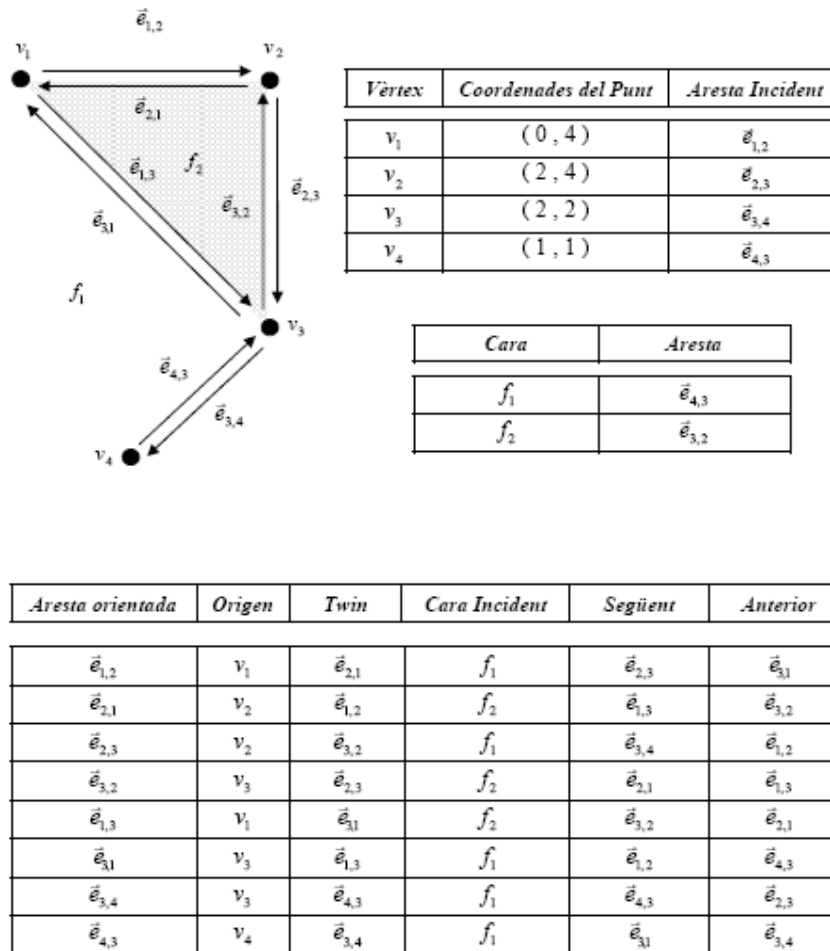


Figura 2.12 : Exemple d'estructura DCEL

2.3 Treballs anteriors

Tal i com ja hem anunciat en l'apartat d'introducció (2.1) d'aquest capítol, al llarg dels anys diversos autors han intentat solucionar el problema dels camins mínims, obtenint resultats diversos i, intentant sempre aconseguir el menor temps d'execució possible. Les referències bibliogràfiques a les quals ens podem remetre per tal d'obtenir informació més precisa sobre el que van aportar els diferents autors són les següents: [MMP87] , [SSKGH05], [CH96], [KO00] i [Kapoor93].

Per a realitzar aquest projecte, hem decidit implementar l'algorisme de trobar camins mínims en un terreny basant-nos en les directrius que s'expliquen en la referència [SSKGH05] sobre l'algorisme d'ordre $O(n^2 \log n)$ proposat per Mitchell [MMP87]. Aquest algorisme ens produeix una solució exacta per al problema “una seu, totes les destinacions” d'una triangulació, que nosaltres ampliarem fent que es pugui obtenir també solució per a un conjunt de seus.

2.4 Preliminars

Després d'haver vist tot el treball anterior que s'ha fet al llarg del temps sobre la problemàtica que ens envolta i quina serà la idea que utilitzarem per tal d'implementar l'algorisme, anem a comentar la idea general i el concepte de dos dels elements més importants que s'utilitzaran en el nostre projecte i que seran importants per tal de portar a terme i poder resoldre amb èxit els problemes de proximitat en terrenys. Aquests conceptes són **l'algorisme de Dijkstra** i els **Diagrames de Voronoi**.

2.4.1 Algorisme de Dijkstra

L'algorisme de Dijkstra, altrament anomenat també algorisme de camins mínims, és un algorisme que fou descobert per primera vegada al 1959 per *Edsger Dijkstra*. És un algorisme que ens serveix per a determinar el camí més curt donat un vèrtex origen a la resta de vèrtexs en un graf dirigit i amb pesos a cara aresta. Així doncs, aplicat a la nostra problemàtica, podem dir que aquest algorisme ens permet solucionar el problema del camí més curt a una seu per a un graf dirigit amb arestes amb pesos positius.

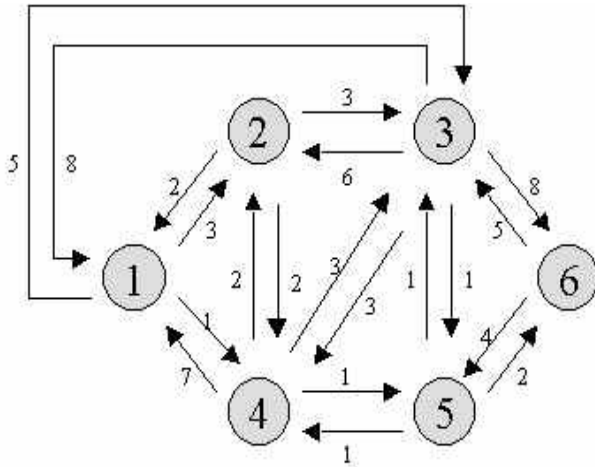


Figura 2.13 : Exemple de l'Algorisme de Dijkstra

L'entrada de l'algorisme consisteix en un graf dirigit G amb pesos a les arestes i un vèrtex seu s del graf G . Els pesos de les arestes es donen mitjançant una funció, on el cost $w(u,v)$ és el cost positiu d'anar del vèrtex u al vèrtex v . L'algorisme ens troba el camí d'anar d'un vèrtex s a un vèrtex t amb menor cost (per exemple, si tractéssim amb distàncies, ens donaria el de distància mínima). També pot ésser utilitzat per tal de trobar el camí més curt d'un vèrtex s a la resta de vèrtexs del graf G .

L'algorisme treballa guardant per a cada vèrtex v el cost $d(v)$ del camí més curt trobat fins al moment entre s i v , és a dir, el camí més curt fet fins al moment per tal d'arribar a v . Inicialment, aquest valor és 0 per al vèrtex seu s ($d(s)=0$) i infinit per a la resta de vèrtexs, representant que de moment encara no coneixem cap camí per anar als altres vèrtexs. Quant l'algorisme acaba, $d(v)$ és el cost del camí més curt de s a v , o infinit si no existeix cap camí entre els dos vèrtexs. L'operació bàsica de l'algorisme de **Dijkstra** és la següent: si hi ha una aresta que va de u a v , llavors el camí conegut més curt d'anar de s a u ($d(u)$) és afegit al camí d'anar de s a v afegint l'aresta (u,v) al final. Aquest camí tindrà la llargada que ja tenia més el pes de l'aresta (u,v) . Si aquest resultat és inferior que la distància actual $d(v)$, reemplacem aquest valor de $d(v)$ amb el nou valor. D'aquesta manera, ens assegurem que sempre estiguem tractant amb camins mínims.

Finalment comentar que el temps de l'algorisme de *Dijkstra* és de l'ordre de $O(V^2)$, essent V el nombre total de vèrtexs, i que no funciona en grafs amb arestes amb pes negatiu.

2.4.2 Diagrames de Voronoi

En el camp de la Geometria Computacional i en el dels Sistemes d'Informació Geogràfica (GIS), la proximitat de Voronoi sobre un conjunt d'objectes donats es sol representar utilitzant els **Diagrames de Voronoi**.

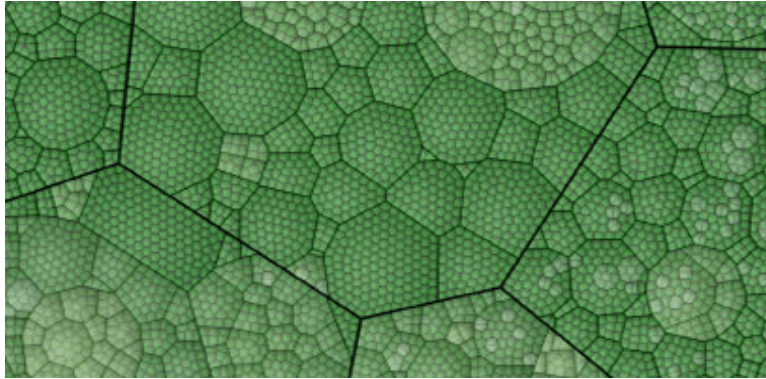


Figura 2.14 : Exemple de Diagrama de Voronoi

El **Diagrama de Voronoi** d'un conjunt de punts de R^d és un concepte fonamental de la geometria computacional les propietats del qual han estat àmpliament estudiades al llarg dels anys per diversos autors. Així doncs, d'aquests Diagrames ja en parlava Descartes, i posteriorment altres autors com Dirichlet, Voronoi i Thiessen van aprofundir encara més en l'estudi d'aquests diagrames. Si considerem S com a un conjunt de punts en el pla, altrament anomenats *seus*, podem definir informalment el *Diagrama de Voronoi de S* com una subdivisió en el pla associant cada *seu* del nostre sistema d'informació amb la seva *regió de Voronoi* corresponent. La **regió de Voronoi** d'una *seu* s_i , està formada per la regió de punts de R^d que estan més a prop de la *seu* s_i que de qualsevol altra *seu* s_j . Comentar el fet de que quan estem parlant de termes de proximitat, ens basem en la *distància Euclidiana*. Així doncs, podem dir que el Diagrama de Voronoi és la subdivisió produïda per les regions de Voronoi.

Els Diagrames de Voronoi ens ofereixen moltes utilitats. Entre les principals hi trobem:

- Posicionament de torres en telefonia mòbil: La regió de Voronoi de cadascuna de les torres ens servirà per determinar quins telèfons haurien de realitzar la connexió a través d'aquesta.
- Control aeri: El Diagrama de Voronoi de cada centre de control ens determinaria la zona d'espai aeri a controlar per aquella estació.
- Distribució de serveis públics (hospitals, centres comercials, ...): La ubicació d'aquests establiments, teòricament hauria d'ésser la que tingués la major àrea de regió de Voronoi respecte a la resta d'establiments del mateix tipus, per tal que d'aquesta manera pugui augmentar hipotèticament la clientela. Un exemple pràctic a la vida real

d'això, si estiguéssim parlant de la ubicació d'hospitals, seria el fet que a un malalt sempre li interessaria anar a l'hospital més proper al seu domicili segons la distància euclidiana. Per saber a quin hospital ha d'anar, només hauria de mirar el Diagrama de Voronoi dels punts que indiquen les posicions dels hospitals i comprovar en quina regió d'aquestes es troba el seu domicili.

El concepte del **Diagrama de Voronoi** s'ha anat fent extensible en diverses direccions per tal de facilitar les aplicacions pràctiques. Així doncs, per exemple també es poden trobar exemples pràctics agafant seus de diferents formes o natures, associant un cert pes a cada seu, o utilitzant funcions de distància individuals a les seus.

2.5 Algorisme exacte de trobar camins mínims

2.5.1 Descripció de l'algorisme

L'algorisme MMP de Mitchell [MMP87] ens calcula el camp (domini) de distància geodèsica en una superfície polièdrica P d'una seu v_s . Els camins mínims són visualitzats com a rajos que surten d'un vèrtex seu v_s en totes les direccions tangents de P al vèrtex v_s . La idea bàsica consisteix en agrupar els camins mínims d'una manera que puguin ésser parametritzats de forma automàtica. Amb aquest objectiu, els costats es parteixen en una sèrie d'interval·ls anomenats **finestres**. Els camins més curts que conté una finestra vénen a través de les mateixes cares i passen per els mateixos vèrtexs. Les finestres són codificades com a una 6-tupla $(b_0, b_1, d_0, d_1, \sigma, \tau)$ i propagades a través de les cares adjacents seguint el mètode de **Dijkstra**. Al llarg d'aquest apartat 2.5 entrarem amb més profunditat amb cadascun dels passos que segueix el mètode i la seva implementació. Per a més detalls, consultar la referència [SSKGH05].

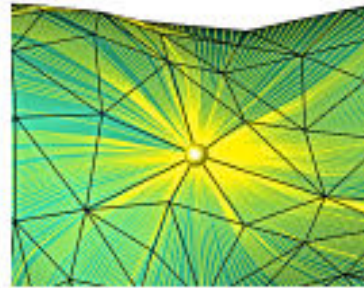


Figura 2.15 : Exemple d'un vèrtex seu

A partir d'ara, una seu general s d'una superfície polièdrica P farà referència a un punt (de moment no realitzarem el càlcul per a segments, polígons, poligonals, ...). També hem de comentar el fet que a part de la seu original s , també tindrem altres seus puntuals o pseudosources p , que seran els que es formaran quan realitzem la propagació de finestres. Aquests *pseudosources* sempre seran vèrtexs de la triangulació del terreny. Es pot realitzar la caracterització dels camins mínims en superfícies triangulars no convexes amb tres propietats:

- 1 → Dins d'una cara d'un triangle el camí més curt és una línia recta
- 2 → Quant travessa un costat, el camí més curt es correspon a una línia recta quan dues cares adjacents es fan coplanars
- 3 → Els camins mínims només poden passar per un vèrtex quan aquest és un vèrtex hiperbòlic o de sella ($>2PI$)

La funció de distància del camí més curt definit per un punt seu s , és una funció D_s tal que per a cada punt q de la superfície polièdrica P , $D_s(q)$ és la llargada del camí més curt de q a la seu s . Així doncs, el camí més curt de la seu s a qualsevol altra punt destí q tindrà la caracterització que acabem de descriure.

Donada una seu puntual p continguda dins del triangle t , utilitzarem l'algorisme desenvolupat en la referència [SSKGH05] per a una seu, només fent alguns canvis en el procés d'inicialització. Quan p sigui interior a t , crearem noves finestres codificant la funció de distància als costats de t . Si p està en un costat e de t i possiblement també en un costat del triangle adjacent t' , crearem dues finestres en el costat e anant de la seu p a cadascun dels extrems de e respectivament i també posarem finestres en els costats restants del triangle t i possiblement en els costats del triangle adjacent t' .

2.5.2 Estructura de les finestres

Tal i com ja hem comentat, el nostre algorisme treballarà principalment amb **finestres**. El camí més curt d'una seu v_s a un punt p del costat e que no passa per cap vèrtex, esdevé una línia recta quan totes les cares que travessa són desplegades en un pla comú. Tots els punts del costat e que en el seu desplegament són ajuntats amb la seu v_s mitjançant línies rectes, defineixen una finestra w en el costat e , la qual és una part associada o connectada al costat e . Aquests camins formen un seguit de rajos emergents de v_s que van a través de la finestra w (veure *Figura 2.16 (a)*).

Aquest conjunt de camins es guarden en una finestra de la següent forma: dos coordenades paramètriques **b0** i **b1** en el rang $[0,|e|]$ (on $|e|$ és la llargada del costat e) que mesuren la distància dels punts extrems de la finestra a l'origen del costat e , dos distàncies **d0** i **d1** que mesuren la distància dels extrems de la finestra a la seu s original, un paràmetre **sigma**, que ens indicarà el pes que té la finestra (serà la distància de la seu original s a la seu puntual o *pseudosource* p que s'hagi format al realitzar la propagació de finestres), que ens servirà per tal d'ordenar-los a una cua de prioritat i així establir un ordre de propagació, i una direcció **tau**, que especifica el sentit del costat on la seu està posicionada. A partir d'aquests paràmetres i del costat, la seu serà fàcilment localitzable i el seu camp de distància serà recuperat. (veure *Figura 2.16 (b)*).

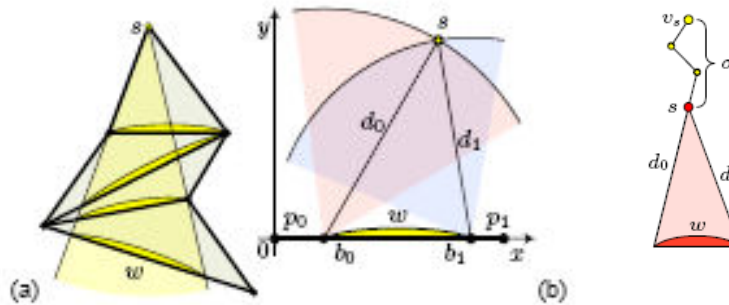


Figura 2.16 : (a) Rajos emergents de la seu s a través de la finestra w al llarg de la tira de triangles oberts. (b) La posició de s a partir dels paràmetres de la finestra. (c) Finestra emergent d'un pseudosource s a distància σ de la seu original

FINESTRA = [b0 , b1 , d0 , d1 , sigma , tau]

2.5.3 Propagació i intersecció de finestres

Un cop ja hem vist quina serà l'estructura de les finestres, anem a veure amb què consistirà la propagació d'aquestes. Haurem de propagar la funció de distància D_s codificada en una finestra d'un costat e al següent triangle adjacent t , creant noves finestres en els dos costats oposats de t . Aquestes noves finestres seran finestres possibles, ja que pot ésser que intersequin o es solapin amb finestres creades anteriorment. En conseqüència, haurem d'intersecar les finestres possibles amb les anteriorment creades i determinar la mínima funció de distància combinada. És a dir, quan ens trobem que això passi, les finestres seran retallades mantenint el mínim camp de distància.

Donada una finestra d'un costat e , nosaltres propagarem la funció D_s calculant com el feix de rajos rectes que representen geodèsiques associades a la finestra w s'estén a través d'un o més triangles desplegats t adjacents a e . Les noves finestres possibles poden ésser creades en els costats oposats del triangle t (veure *Figura 2.17 (a) i (b)*). Per tal de propagar la finestra w del costat e a la cara f , busquem la posició del pseudosource s en el pla definit per la cara f . Llavors, els rajos emergents des de la seu a través dels extrems de w seran considerats per tal de determinar l'interval de la nova finestra (b_0', b_1') en el costat e' . Les noves distàncies d_0' i d_1' respecte a la seu també seran calculades, mentre que el paràmetre σ no ens canviarà.

Quant la finestra w és adjacent a un vèrtex frontera o de sella (que serà sempre un vèrtex de la triangulació), les geodèsiques poden anar a través d'ella, per això, actua com a un nou pseudosource i ens genera un nou pseudosource que és tractat com una nova "seu puntual", amb $\sigma = d_i$, on $i=0$ ó $i=1$ (veure triangle amb color de la *Figura 2.17 (c)*). Finalment comentar que per tal de propagar els **PseudoSources** (vèrtexs del terreny que siguin hiperbòlics), el que farem és inserir finestres a tots els costats dels triangles adjacents del PseudoSource en qüestió.

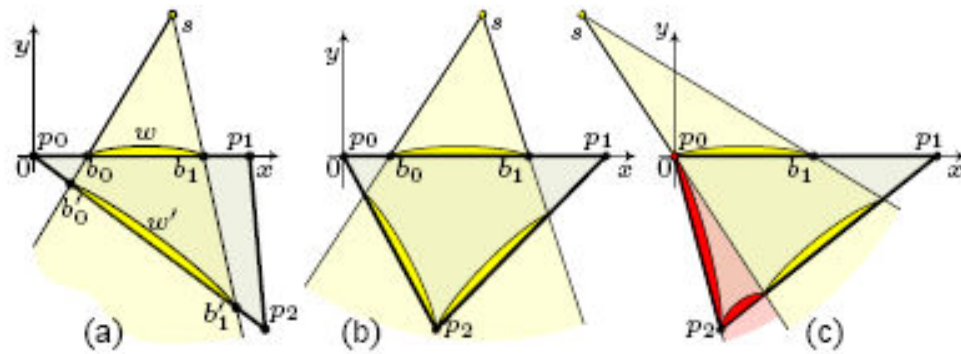


Figura 2.17 : (a,b) Exemples de propagació de finestres que ens formen : (a) una nova finestra a un costat ; (b) dues noves finestres en dos costats diferents. (c) Es forma una nova finestra i un nou vèrtex pseudosource v .

Després de la propagació, les noves finestres possibles w' en el costat e' poden solapar-se o intersecar amb finestres creades anteriorment. Sigui w una finestra anteriorment creada que es solapa amb la finestra w' , llavors haurem d'ésser capaços de decidir quina finestra defineix la mínima distància en el sub-segment solapat $\delta = [b_0, b_1] \cap [b_0', b_1']$. Per tal d'obtenir de forma correcta la finestra, hem de calcular el punt en δ on les dos funcions de distància coincideixen. Llavors, les finestres seran retallades mantenint el camp de distància mínima. Descartarem les geodèsiques guardades en les finestres w i w' que no puguin ésser camins mínims. En la figura següent es pot observar gràficament el que es realitza quan dues finestres intersequen o es solapen.

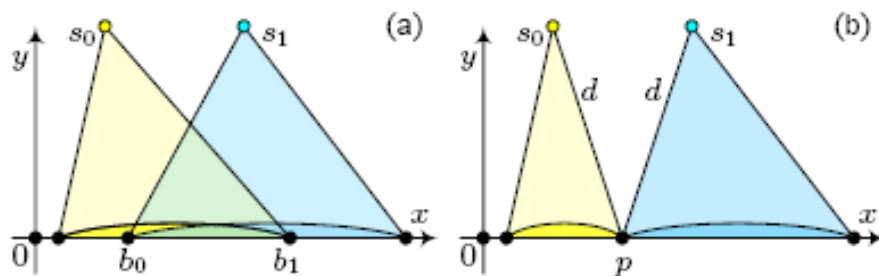


Figura 2.18 : (a) Dues finestres w_0 i w_1 que es solapen (amb els seus respectius pseudosources s_0 i s_1) i la seva intersecció $[b_0, b_1]$. (b) La formació de dues finestres disjunts a partir de dues finestres que es solapaven, amb els nous intervals $b_0' - b_1'$ per totes dues i amb les noves distàncies d .

2.5.4 Reconstrucció del camí mínim (Backtracing)

Un cop hem vist com seran les finestres que guardarem en els costats del terreny, com es realitzarà la propagació de finestres al llarg del terreny i què farem quan ens trobem amb finestres que intersequin o es solapin, anem a veure com ho farem per tal de realitzar l'algorisme que ens permeti reconstruir el camí mínim. L'algorisme comença amb un pas d'inicialització que ens

cobreix amb finestres els costats dels triangles que contenen la seu original (seu) v_s . Aquestes finestres creades són inserides en una cua de prioritat amb un pes igual a la distància respecte a la seu v_s . Utilitzant un mètode continu de *Dijkstra*, la primera finestra de la cua de prioritat (la que tingui el pes més petit utilitzant com a $pes = \min(d_0, d_1) + \sigma$) és seleccionada, propagada i eliminada. Després, una vegada s'hagin inserit noves finestres als costats adjacents o bé s'hagin actualitzat finestres que es solapaven, aquestes també seran inserides a la cua de prioritat amb el seu respectiu pes. Tots aquests passos s'aniran realitzant fins que la cua de prioritat sigui buida. Cal remarcar el fet que quant ens trobem amb un solapament de finestres, aquestes poden ésser modificades o esborrades de la cua de prioritat, fet que s'ha d'actualitzar immediatament en la cua.

Després de tenir finestres al llarg de tot el terreny com a conseqüència d'haver propagat el camp de distància a tot el terreny (superfície polièdrica), el camí més curt des de qualsevol punt q d'un triangle t respecte a una seu donada s es pot obtenir fàcilment utilitzant una tècnica de **backtracing**. Primer de tot, escollim la finestra w' en els costats del triangle t que defineix la mínima distància respecte el punt q . Remarquem el fet que la distància al punt q donada per una finestra w és $d_w(q) = d_w(q') + |q - q'|$ essent q' en la finestra w el punt més proper a q i d_w la funció de distància generada per la finestra w . Des de la finestra w' , saltarem a un triangle adjacent t' utilitzant la direcció τ guardada en la finestra. Anirem saltant a triangles adjacents fins que arribem al triangle que conté la seu original s . Llavors, al anar-nos guardant en cada pas el punt per el qual hem passat, solament haurem d'ajuntar aquests punts per tal d'obtenir el camí més curt.

2.6 Terrenys amb més d'una seu

Aquest algorisme que acabem d'explicar, és fàcilment extensible al cas general d'un terreny amb un conjunt de seus. En aquest cas, s'obté una funció de distàncies generalitzada, la qual per a cada punt ens dona el seu camí de distància mínima a la seu més propera. Només ens és necessari canviar el pas d'inicialització, concretament ara generarem finestres per cada seu i les guardarem totes en una única cua de prioritat. D'aquesta manera, propaguem simultàniament el camp de distàncies definit per cada una de les diferents seus. Automàticament podrem obtenir una codificació de la funció de distàncies generalitzada que ens produeix una representació implícita del diagrama de Voronoi del conjunt de seus.

2.6.1 Representació diagrames de Voronoi

En l'apartat 2.4.2 d'aquest capítol ja havíem explicat quin era el concepte teòric i per a què ens servien els **Diagrames de Voronoi**. Recordem que els Diagrames de Voronoi eren la estructura més típica per tal de representar termes de proximitat relacionats amb una sèrie de punts (seus) en la Geometria Computacional.

En ordre d'obtenir el Diagrama de Voronoi d'un conjunt de seus $O=\{s_1, \dots, s_k\}$, utilitzarem l'algorisme que hem fet per tal de calcular la funció de distància a la seu més propera. Això es durà a terme considerant totes les seus en el pas d'inicialització i inicialitzant la cua de prioritat amb totes les finestres. Quant estem treballant amb diferents seus, és interessant conèixer de quina seu prové cada finestra nova que es va creant i inserint. Amb aquest objectiu, guardem un paràmetre extra a cada finestra, anomenat j , que anirà de 1 fins a k (essent k el nombre de seus). Així doncs, per a cada finestra afegirem un nou paràmetre a la tupla que ja havíem definit anteriorment. Aquest paràmetre, doncs, serà l'índex de la seu de la qual la finestra prové.

Per tal de visualitzar una aproximació del Diagrama de Voronoi, calcularem una aproximació del camp de distàncies utilitzant només la distància als vèrtexs del terreny i hardware gràfic. Durant aquest procés, necessitarem guardar a cada vèrtex del terreny la distància a la seu més propera i l'índex d'aquesta seu.

Un cop aconseguim aquesta informació, una aproximació discreta del Diagrama de Voronoi es pot obtenir utilitzant hardware gràfic, fent ús de la inherent interpolació bilineal durant el procés de rasterització. El Diagrama de Voronoi s'obté utilitzant *OpenGL* i agafant els avantatges que ens produeix el Test de Profunditat (*Depth Test*). Anirem pintant un darrera l'altre els camps de distàncies de cada seu mitjançant colors diferents. Si pintem el mínim valor de la coordenada z , obtindrem la mínima distància de cada punt respecte al conjunt de seus. Quant totes les seus hagin estat tractades, la imatge obtinguda en el "*frame buffer*" és el Diagrama de Voronoi, i els valors guardats en el "*buffer*" de profunditat (*z-buffer*) és el camp de distància associat al conjunt de seus.

Per tal de representar la funció de distància i obtenir el Diagrama de Voronoi, pintarem les cares del terreny amb una funció aproximada de distància amb el color associat a la seu que defineix la funció de distància. Podrem diferenciar dos casos diferents:

- **Cares amb els 3 vèrtexs amb la mateixa seu propera:** La distància d'un punt p en una cara amb 3 vèrtexs v_i amb distàncies d_i al punt s es pot obtenir utilitzar interpolació lineal des de les distàncies d_i . Així doncs, si $\mathbf{p} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + c_3\mathbf{v}_3$ on $c_1 + c_2 + c_3 = 1$, podem aproximar la distància de s a p com $c_1d_1 + c_2d_2 + c_3d_3$. Això pot ésser realitzat a terme senzillament, pintant per a cada triangle del domini de la triangulació corresponent a la cara f un triangle amb els vèrtexs de f , però reemplaçant la coordenada z per la distància d_i normalitzada a $[0,1]$. Una vegada tots els triangles han estat pintats, la triangulació de Delaunay ha estat estesa de forma aproximada a tots els punts del domini.

- **Cares amb vèrtexs amb diferents seus properes:** Per a cada seu diferent s_k pintarem un camp de distàncies aproximat en la cara associada a la seu s_k . Assumint que v_0 és a distància d de la seva seu més propera s_k , per a cada vèrtex v_i de la cara el qual la seva seu més propera no és s_k , aproximarem la distància del vèrtex v_i a la seu s_k com $d + d(v_0, v_i)$. Per això, pintarem la cara almenys 3 vegades utilitzant una aproximació de camp de distàncies diferent donada cada vegada per a una seu diferent.

2.6.2 Anàlisi de complexitat

La funció de distància per a un conjunt de seus en una superfície polièdrica no convexa P amb obstacles poligonals, representada com a una triangulació formada per n triangles, es pot obtenir de forma intrínseca en ordre $O(N^2 \log N)$ en quant a temps i $O(N^2)$ en quant a espai, on N és el màxim de n i el nombre de seus. Això ha estat provat i demostrat a la referència [MMP87].

Una consulta de distància demanant la llargada del camí més curt d'un punt de P a la seva seu més propera, pot ésser solucionat mitjançant mètodes estàndards en temps $O(\log N)$. Finalment, el camí més curt pot ésser calculat en un temps addicional $O(k)$, on k és el nombre de triangles travessats per el camí.

Fonaments Pràctics

2.7 Introducció

En aquesta part del capítol explicaré les eines, llibreries, software, etc. que he hagut de provar i entendre per llavors facilitar la implementació de la part teòrica. Així doncs, es mostraran les eines que han permès la implementació de l'aplicació així com els programes necessaris per tal de construir la documentació corresponent. Comentar el fet que algunes d'aquestes eines jo gairebé mai les havia tractat (o hi havia tractat molt per sobre), per la qual cosa en l'inici del projecte vaig haver de fer durant un cert període de temps tot un procés d'aprenentatge per tal de familiaritzar-m'hi i entendre més profundament el seu funcionament.

2.8 La llibreria Qt (KDE)



Qt és una llibreria escrita en **C++**. El podríem definir com un *toolkit* per al desenvolupament d'aplicacions en mode gràfic que utilitza el llenguatge de programació C++. Està format per una biblioteca de classes implementades en C++ i un conjunt d'eines que permeten construir interfícies

gràfiques d'usuari.

Les Qt han estat creades i mantingudes per **Trolltech**, una companyia de software de nivell internacional. No és tan sols una llibreria de classes, sinó que destaca per ésser una llibreria molt completa i multiplataforma, que permet crear interfícies gràfiques d'usuari (GUI) amb facilitat ja que disposa d'una excel·lent documentació accessible de forma gratuïta des de la pàgina web de *Trolltech*. La filosofia de les Qt és “*write once, compile anywhere*”, que vol dir: “*escriu un cop, compila en qualsevol lloc*”. Aquesta dita fa referència a la principal característica de les Qt, el fet que sigui multiplataforma, de manera que un mateix codi pot funcionar correctament en qualsevol plataforma simplement compilant l'aplicació sense haver de fer canvis en el codi.

TROLLTECH™

A més a més, la llibreria **Qt** està disponible per un gran nombre de sistemes operatius com les plataformes de Linux, Mac OS X, Windows, Solaris,

HP-AUX i moltes altres versions de Unix X11. La seva gran portabilitat no serveix com a inconvenient del programador, ja que el codi utilitzat en qualsevol de les plataformes és exactament el mateix. Addicionalment és possible obtenir una versió Qt per a sistemes Linux immersos (*embedded Linux*). La versió per Unix és de lliure distribució sempre i quan no se'n comercialitzi el programa que la faci servir. La versió per Windows és plenament comercial però existeix una versió gratuïta si no és per ús comercial.

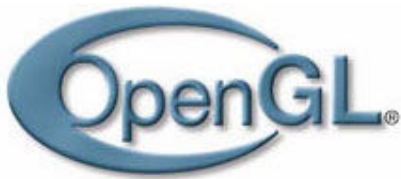
Tal i com hem comentat anteriorment, Qt utilitza el llenguatge de programació C++, però també permet utilitzar altres llenguatges com C, Python i Perl. A més a més, disposa de suport per tal d'accedir a bases de dades sense interfície gràfica mitjançant SQL, suport per XML i una excel·lent API per manipular fitxers. Cal dir que existeix també la variant *QSA (Qt Scripts for Applications)* que permet integrar scripts en les aplicacions creades amb Qt.

Qt destaca per ser la llibreria base de l'entorn de finestres **KDE**. Actualment existeixen varies eines per generar la interfície de forma visual. La pròpia llibreria en proporciona una, el **Qt Designer**, que posteriorment explicarem amb més detall. Comentar també que la llibreria Qt proveeix de molts de *widgets* i en permet la creació de nous. Com a principals defectes té el fet de que no sigui de lliure distribució per a tots els sistemes operatius i que el codi generat és poc flexible i difícil de modificar.

Les llibreries aportades per Qt (*versió qt-x11-3.3.3*) han estat útils en la realització del projecte per tal de crear tota la interfície gràfica de l'aplicació. Les classes obtingudes de Qt han permès dissenyar la finestra del programa amb cadascun dels diferents tipus de controls que hi apareixen. A més a més, el fet que utilitzi el llenguatge C++ fa que sigui totalment compatible amb altres llibreries implementades amb aquest mateix llenguatge. D'altra banda, gràcies al fet que proporcioni eines per a la programació amb events (clic del ratolí, tecles premudes, timers, signals-slots, ...) ha fet que hagi estat molt més fàcil la tasca d'interacció amb l'usuari.

Podem trobar més informació de les llibreries **Qt** a la seva pàgina Web: <http://www.trolltech.com>

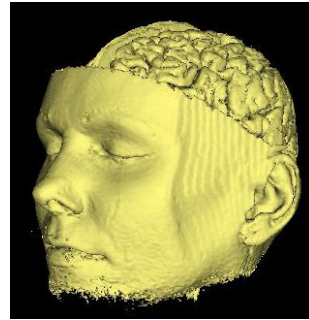
2.9 La llibreria OpenGL



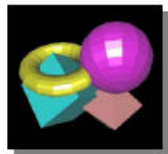
OpenGL (*Open Graphics Library*) és una interfície de software per a hardware gràfic, és a dir, es tracta d'una API portable que permet desenvolupar aplicacions amb gràfics interactius 2D i 3D. Ha estat creada per *Silicon Graphics* i aprofita les característiques del hardware gràfic de la màquina. La interfície consisteix en un elevat nombre de procediments i funcions que permeten al programador especificar els objectes i les operacions necessàries per produir imatges gràfiques de gran qualitat.

L'API està formada per un gran nombre de procediments i funcions de renderitzat, mapeig de textures, efectes especials i de visualització. És totalment multiplataforma, de manera que dóna suport per a la majoria de sistemes. Normalment són utilitzades per a crear imatges en color que contenen objectes tridimensionals. Disposa d'un gran nombre de comandes que afecten a les operacions realitzades sobre hardware gràfic. Qualsevol targeta gràfica amb accelerador 3D conté implementada, per hardware, la biblioteca **OpenGL**. En el cas que no la contingui, pot ésser instal·lada per software a través de la llibreria *MESA* (port de les OpenGL). Tot seguit podem veure algunes de les principals característiques que posseeix *OpenGL*:

- Qualitat visual elevada així com un alt rendiment. Per aquest motiu, **OpenGL** és utilitzada en àrees com animació 3D, CAD, simulació visual, realitat virtual, imatges mèdiques, ... on en totes elles és necessari el màxim rendiment amb la millor qualitat visual possible.



- Avantatges dirigits als programadors: OpenGL és un estàndard en l'àrea de gràfics per computador, és estable, evoluciona contínuament, és portable, escalable, fàcil de fer servir i disposa d'una àmplia documentació.



- OpenGL opera directament sobre la informació d'una imatge a través de primitives geomètriques.
- Simplifica la tasca de desenvolupament del programari. Dóna al programador accés a primitives geomètriques i d'imatge, display lists, transformació dels models, il·luminació, textures, anti-aliasing, suavitzat, ... que permeten al programador disminuir el temps de desenvolupament. Dóna suport a diferents llenguatges de programació, de manera que poden ésser utilitzades en aplicacions desenvolupades amb els llenguatges de programació C, C++, Java, Fortran i Ada.
- Les biblioteques de OpenGL es troben disponibles per a les plataformes de Unix/Linux, Windows, BeOS, Mac OS X. OpenGL és totalment independent dels protocols de xarxa i de les topologies. En el sistema operatiu Unix n'existeix una de lliure distribució anomenada *Mesa*. El sistema operatiu Windows la porta incorporada de sèrie.

- OpenGL és flexible, de manera que pot utilitzar-se per realitzar operacions senzilles o per crear elements complexos. Pot funcionar en un PC o en un supercomputador, de manera que un programador no s'haurà de preocupar de la implementació d'OpenGL per a la seva plataforma de desenvolupament, ja que els resultats seran consistents.



La interfície que s'utilitza en aquest projecte, utilitza la llibreria *OpenGL* a l'hora de programar tota la part de visualització del programari, des de l'**Editor**, on es representaran elements en 2D, fins al **Visor**, on es mostren elements en 3D.

Podem trobar més informació de les llibreries **OpenGL** a la seva pàgina Web: <http://www.opengl.org>

2.10 La llibreria CGAL



Les llibreries **CGAL** (*Computational Geometry Algorithms Library*) són unes llibreries de domini públic que han estat desenvolupades gràcies a l'esforç col·lectiu de diverses universitats i empreses europees que treballen en *Geometria Computacional*, per tal de desenvolupar una llibreria d'estructures de dades i algorismes geomètrics robusta, eficient i fàcil d'utilitzar, en el llenguatge de programació C++.

Avui en dia, **CGAL** s'ha convertit en un estàndard universal per a aquesta classe de software. Bàsicament, les llibreries CGAL contenen:

- Primitives geomètriques bàsiques, com poden ésser punts, vectors, rectes, posicions relatives de punts i operacions tals com interseccions i càlculs de distàncies.
- Una col·lecció d'estructures de dades i algorismes estàndard, tals com l'envolvent convexa, la triangulació de Delaunay, poliedres, estructures de consulta multidimensional, ...

- Interfícies amb altres paquets, principalment per a visualització, entrada/sortida o aritmètica.

Podem trobar més informació de les llibreries **CGAL** a la seva pàgina Web: <http://www.cgal.org/>

2.11 KDevelop



El projecte **KDevelop** va sorgir al 1998 amb la finalitat de desenvolupar un **IDE** (*Integrated Development Environment*) gratuït, complet i fàcil d'utilitzar per **KDE**. Permet desenvolupar aplicacions principalment en C i

C++ per a plataformes GNU/Linux i altres sistemes operatius compatibles amb Unix. KDevelop es troba públicament disponible sota la *GPL* (*Llicència Pública GNU*).

KDevelop no inclou un compilador, però utilitza la col·lecció de compiladors de GNU (GCC) o altres compiladors per tal de produir codi executable. Actualment dona suport a una gran nombre de llenguatges de programació com són Ada, Bash, Fortran, Java, Pascal, Perl, Python, Ruby, SQL, C i C++.

Permet doncs, desenvolupar aplicacions de *KDE*, *Qt* o terminal a través de diverses eines. Utilitza la tecnologia *KPart*, la qual li permet utilitzar un editor de text extern, immers dins *l'IDE* com si fos un component. L'editor que utilitza per defecte és el *KDE Advanced Text Editor* o el *KATE*. Les principals característiques que aquesta eina ofereix als seus usuaris són les següents:

- Editor de codi font amb remarcats de sintaxi i identificació automàtica.
- Gestió de projectes de diferents tipus. Utilitza les eines *Automake*, *qmake* per a projectes basats en les Qt i *Ant* per a projectes basats en Java
- Navegador de classes
- Serveix com a interfície per la col·laboració de compiladors de GNU (GCC).
- Serveix com a interfície per al debugador de GNU.
- Completa el codi automàticament (només per C i C++)
- Suport per l'aplicació de generació de documentació *Doxygen*

- Suport per a les aplicacions de control de versions CVS i Subversion.
- Disposa de “*syntax highlighting*” (permet destacar el codi mitjançant colors o tipus de lletra), editor de diàlegs visual, depurador integrat, plantilles per aplicacions, suport per traduccions, editor de icones, accés ràpid a la documentació, generació automàtica d'executables, ...

KDevelop està basat en una arquitectura de connectors (plugins). Per aquest motiu, si algun programador realitza algun canvi, només caldrà que compili el connector. A més a més, no depèn del llenguatge de programació o del sistema on s'executarà l'aplicació, donant així suport a un gran nombre de tecnologies tals com Qt, GTK+ i wxWidgets.

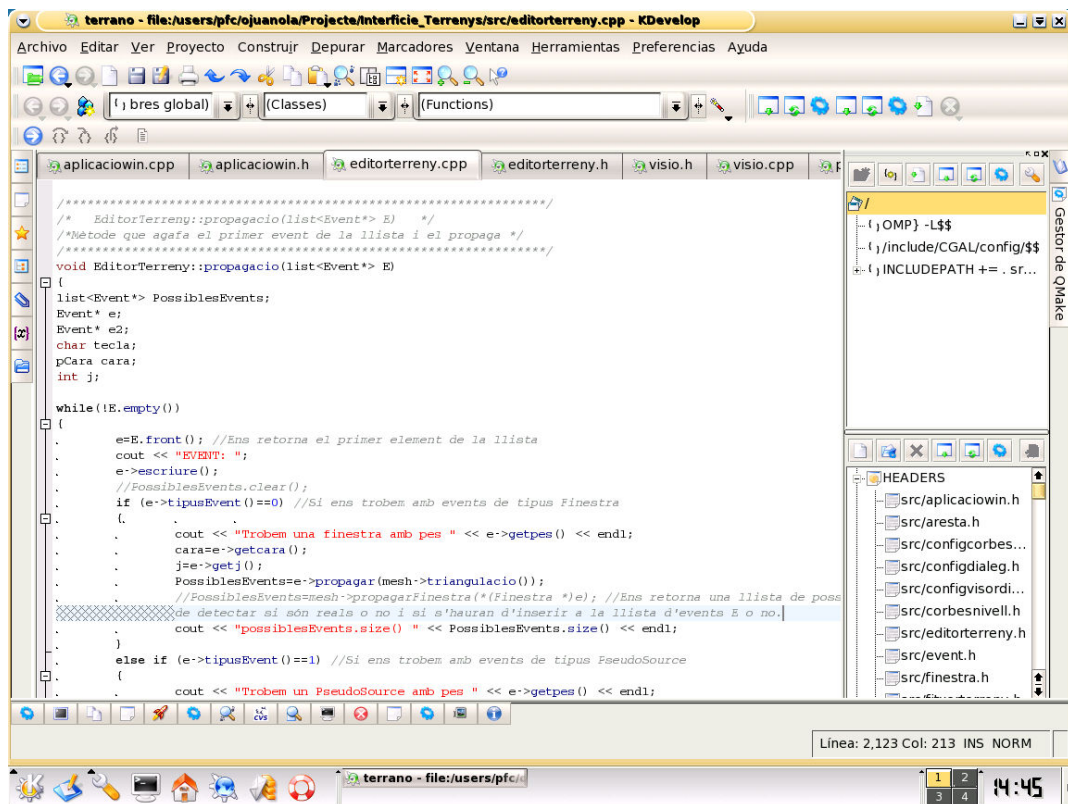


Figura 2.19 : Exemple de la interfície del *KDevelop*

Per tal de programar l'aplicació d'aquest projecte, s'utilitzarà l'eina **KDevelop** (versió 3.0.2 utilitzant KDE 3.2.1) com a editor del codi font. Gràcies a les seves funcionalitats podem construir el programari amb molta més fluïdesa i comoditat que si haguéssim d'utilitzar qualsevol altre editor de text. Una de les funcionalitats de KDevelop serà la de fer d'interfície per les eines de compilació (gmake, g++, ...) de manera que no haurem d'escriure les comandes, sinó que simplement activant un botó es realitzarà l'operació de forma transparent. Fins i tot podrem visualitzar els missatges d'error provocats a l'hora de construir l'aplicació i posteriorment localitzar-los amb un simple clic

de ratolí sobre aquests missatges. L'execució de l'aplicació també és possible realitzar-la mitjançant un sol clic a un botó, sempre i quan l'aplicació hagi compilat correctament.

Totes aquestes funcionalitats, juntament amb les característiques anteriorment esmentades, fan que l'eina KDevelop sigui molt útil per al desenvolupament del programari amb una plataforma Unix/Linux de forma còmoda.

Podem trobar més informació sobre l'eina **KDevelop** a la seva pàgina Web: <http://www.kdevelop.org>

2.12 QtDesigner



QtDesigner és una poderosa *GUI (Interfície gràfica d'usuari)* que disposa de totes aquelles eines necessàries per tal que l'usuari pugui construir formularis, diàlegs i aplicacions a través de les llibreries **Qt** que hem explicat anteriorment. Aquesta eina ens permet el desenvolupament ràpid d'interfícies d'usuari d'alt rendiment per a qualsevol tipus de plataforma suportada per les Qt.

Independentment, o bé integrat amb algun *IDE* com *Microsoft Visual Studio .NET*, **QtDesigner** inclou poderoses eines com per exemple el mode de previsualització, suport per a elements bàsics d'interfície, distribució automàtica d'aquests, editor avançat de propietats, ...

QtDesigner fa que la tasca d'edició sigui fàcil gràcies a la visualització del disseny d'interfícies d'usuari avançades. En qualsevol moment es pot generar el codi requerit per reproduir i previsualitzar la interfície que haguem dissenyat, modificant i ajustant el nostre disseny tantes vegades com es vulgui.

A partir d'aquesta eina de disseny podrem distribuir els diferents elements dins la finestra de diàleg tal i com vulguem. Es tracta d'una forma fàcil, ràpida i eficient de construir les interfícies tal i com ho desitgem. A més de dissenyar, també ens permet afegir events i funcionalitats als diversos elements.

En el nostre cas, aquesta eina de disseny (*versió 3.2.1*) ha estat utilitzada per crear algunes de les finestres de diàleg, menús, opcions de menús i icones que hem afegit a l'aplicació original. Per tal de poder obtenir el codi C++ equivalent, un cop finalitzat el disseny, hem de guardar el fitxer amb extensió *.ui* propi del **QtDesigner**. A continuació, des d'un terminal, simplement haurem d'executar les següents comandes:



- `qmake -project` → Ens permetrà generar el fitxer `.pro` equivalent
- `qmake` → Construeix el `makefile` equivalent a partir del `.pro` creat anteriorment.

A continuació, simplement ens caldrà executar el `makefile` amb la comanda **make**, obtenint d'aquesta manera els fitxers de capçalera (`.h`) i d'implementació (`.cpp`).

Podem trobar més informació sobre l'eina **QtDesigner** a la seva pàgina Web: <http://www.trolltech.com/products/qt/designer.html>

2.13 Objecteering UML Modeler



Objecteering UML Modeler és una eina que ens serveix per modelar qualsevol mena de sistema que necessiti programació orientada a objectes o, fins i tot, sistemes que no tinguin res a veure amb software.

Ens proveeix d'unes eines de software que permeten als analistes, dissenyadors i programadors obtenir els avantatges que suposa en enginyeria la utilització de *UML*. Ens permet expressar els requeriments del nostre sistema, la construcció de diagrames/models *UML* complerts i precisos, ens produeix també documentació i estadístiques, així com també disposa de generació de codi automàtica per els llenguatges Java, C, C++, SQL, CORBA i Fortran. A més a més, ens ajuda a realitzar el cercle complet dins de qualsevol programa, és a dir, l'anàlisi de requeriments, el disseny del sistema, la implementació del sistema, els jocs de proves i la obtenció de l'aplicació final.

Així doncs, podem dir que aquesta eina pot simplificar la complexa tasca de desenvolupament de programari ajudant a estructurar pensaments, a clarificar la comunicació i a trobar l'abstracció correcta. Aquesta eina, també disposa d'una interfície intuïtiva que permet a l'usuari alliberar-se i centrar-se simplement en el seu model.

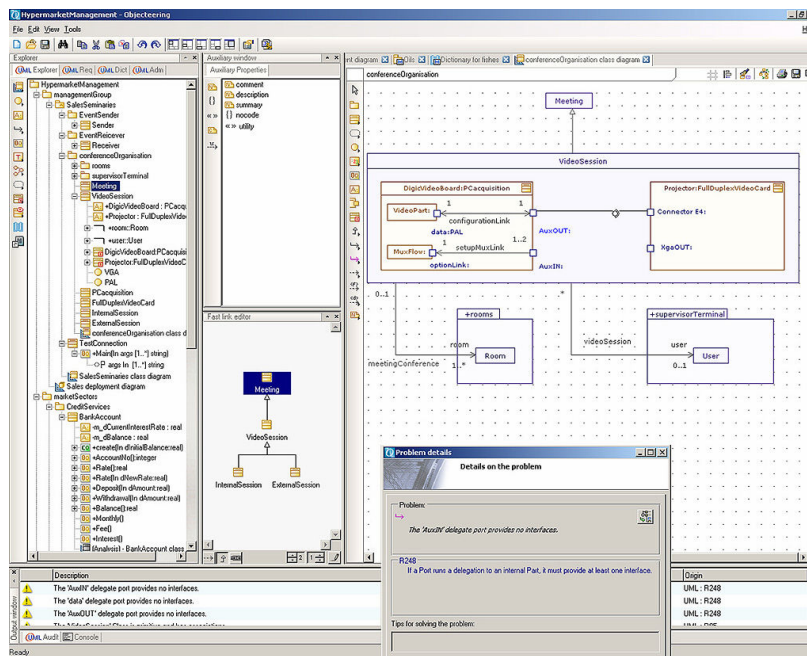


Figura 2.20 : Exemple de la interfície del Objectteering UML Modeler

Podem trobar més informació sobre l'eina **Objectteering UML Modeler** a la seva pàgina Web: <http://www.objectteering.com/>

Capítol 3

Requeriments del Sistema

En aquest capítol de la memòria realitzarem bàsicament l'**anàlisi dels requeriments** que hauria de complir el sistema (tant funcionals com no funcionals). També utilitzarem diverses eines de modelat UML per tal de identificar els actors (usuaris) que interactuaran amb la nostre aplicació, realitzarem diagrames de casos d'ús, fitxes de casos d'ús, diagrames d'activitat, ... Així, sabrem què ens caldrà tant des del punt de vista de software com de hardware per tal que el nostre projecte pugui complir amb els objectius que ens hem marcat en iniciar-lo.

Bàsicament aquest **anàlisi de requeriments** ens permetrà entendre:

- Les persones o elements que interactuaran amb el programari, és a dir, els **actors**. Un actor el podríem definir com a una entitat externa (persona, dispositiu, sistema, ..) que interactua amb un sistema interpretant un determinat rol.
- Els **procediments**, és a dir, les funcionalitats que ha d'oferir la nostra aplicació per tal de poder complir amb els seus objectius.
- El **hardware** o plataforma on funcionarà el sistema (requeriments de targeta gràfica, memòria, espai de disc dur, velocitat, processador, ...)

- El propi **software** o programari que representen les eines que s'utilitzaran a l'hora d'implementar el sistema. En aquest cas caldrà tenir en compte la **portabilitat** cap a altres plataformes de desenvolupament.

Podem classificar els requeriments bàsicament en dos tipus :

- Requeriments Funcionals

Els requeriments funcionals descriuen quins són els serveis o funcionalitats que ens oferirà l'aplicació independentment de la seva implementació. Cada requeriment funcional expressa una relació entre les entrades i sortides del sistema, és a dir, especifica les sortides que s'han de produir a partir d'unes determinades entrades i les operacions necessàries per aconseguir això. També han d'especificar com s'ha de comportar el sistema davant de situacions anormals (entrades invàlides, errors, ...)

- Requeriments No Funcionals

Els requeriments no funcionals ens informen sobre les restriccions que venen imposades pel client o per el propi problema. Aquestes restriccions normalment són considerades com quantificables. Descriuen doncs, els terminis de lliurament, pressupostos, programari de desenvolupament, disponibilitat de recursos, seguretat, interfícies externes, portabilitat, ...

3.1 Requeriments Funcionals

Tal i com ja hem comentat anteriorment, els **requeriments funcionals** d'una aplicació seran aquells serveis o funcionalitats que ens oferirà l'aplicació independentment de la seva implementació, reflectint en tot moment les responsabilitats del programa a construir. A continuació anem a descriure els principals requeriments funcionals de la nostra aplicació:

→ *Inserir / Eliminar elements de distàncies (seus)*

A partir d'aquesta operació, que formaria part de les operacions d'edició del terreny, hem de poder inserir seus en el terreny, que seran principalment punts (posteriorment en un treball futur, es pretén que tot el càlcul de camins òptims es pugui realitzar també treballant amb segments, polígons, poligonals...). L'usuari escollirà la posició de l'Editor 2D on vol inserir la/les seu/s. L'alçada (coordenada Z) no la podrà escollir, ja que farem que ja ens la calculi el programa mateix, de manera que sempre situem els punts a l'alçada del terreny (damunt el terreny mateix).

De la mateixa manera que podem afegir elements de distàncies (seus) a l'Editor 2D del terreny, l'aplicació també ens ha de permetre poder-los eliminar. En el cas d'eliminar elements de distàncies, l'usuari haurà de poder esborrar qualsevol element que hagi introduït prèviament.

→ *Visualitzar el terreny*

Totes les modificacions de seus (insercions i eliminacions) han de poder ésser visualitzats de forma interactiva tant en l'Editor 2D com en el Visor o finestra 3D. Així doncs, tots els canvis d'edició que es produeixin en el terreny, s'han de poder visualitzar.

→ *Calcular la distància geodèsica entre una seu i un punt*

L'aplicació ens haurà de permetre poder calcular la distància geodèsica entre una seu (element de distàncies) i un punt qualsevol del terreny. Entenem com a distància geodèsica la distància més curta que hi ha entre dos punts sobre una superfície definida matemàticament.

→ *Calcular i Visualitzar el camí òptim entre una seu i un punt*

L'aplicació ens haurà de permetre poder calcular quin és el camí (recorregut) òptim, de distància mínima, que hi ha entre un punt del terreny que haguem inserit i la seu (element de distàncies) més propera a aquest, que s'haurà inserit prèviament. Posteriorment, haurem de poder reconstruir aquest camí òptim i visualitzar-lo tant en l'Editor (2D) com en el Visor (3D).

→ *Calcular i visualitzar Diagrames de Voronoi sobre terrenys*

L'aplicació ens haurà de permetre, donat un terreny amb un conjunt de seus inserides (elements de distàncies), trobar/calcular i visualitzar quin és el Diagrama de Voronoi per al terreny amb aquell conjunt de seus.

A més de poder donar resposta a tots aquests requeriments, propis del nostre projecte, també necessitarem utilitzar i/o modificar alguns *altres requeriments* que ja estaven més o menys incorporats a les versions anteriors de la interfície. Així doncs, hi haurà una sèrie de requeriments als quals l'aplicació ja hi donava una resposta positiva que s'han hagut d'utilitzar o s'han hagut de modificar lleugerament o notablement per tal d'adequar-los a la funció que volíem que fessin en el nostre projecte. Tot seguit els anem a comentar:

- **Crear un nou terreny**

El programa ens ofereix la possibilitat de crear un terreny buit, llest per poder ésser generat i/o editat.

- **Generar terrenys**

L'aplicació ens ha de permetre poder generar terrenys, és a dir, triangular un núvol de punts o un *PSLG*. Per tal de generar un terreny, tenim diverses opcions:

- Carregar les dades d'un terreny generat prèviament i emmagatzemat en un format compatible per tal de poder obrir-lo de nou.
- Que l'usuari generi un terreny dibuixant-lo ell mateix escollint les posicions de cadascun dels vèrtexs de la triangulació així com la seva alçada (component Z)
- Generar un terreny de forma *pseudo-aleatòria* a partir d'un determinat algorisme i d'una sèrie de paràmetres introduïts per l'usuari.

- **Editar terrenys**

L'aplicació ens ha de permetre poder editar un terreny. Totes les funcionalitats d'edició seran efectuades en dos dimensions. Principalment trobarem les següents opcions d'edició:

→ ***Afegir / Eliminar elements restringits***

A partir d'aquesta operació hem de poder inserir en el terreny qualsevol mena d'element restringit: punts, segments, poligonals o polígons. El fet d'afegir un element en el terreny implica una retriangulació de la malla anteriorment existent. Com a producte d'aquesta retriangulació l'element restringit queda incorporat dins de la malla. En el nostre cas, sempre que l'utilitzem serà per afegir-hi principalment punts. L'usuari haurà d'escollir la posició on vol inserir-lo i l'alçada corresponent (coordenada z).

De la mateixa manera que podem afegir elements restringits, l'aplicació també ens ha de permetre eliminar-los. En aquest cas, l'usuari haurà de poder esborrar qualsevol dels elements restringits prèviament afegits (ja siguin punts, segments, polígons o poligonals).

→ *Modificar alçada*

El programa ens ha de permetre la modificació de les alçades dels punts de la malla, ja siguin punts generats de forma aleatòria al crear el terreny o elements restringits afegits per l'usuari. En el cas dels elements de distàncies (seus) i dels punts de camí mínim, hem dit que no es disposarà d'aquesta possibilitat ja que ja ens calcularà l'alçada directament i ens posarà la seu o el punt sobre el terreny.

→ *Obtenir informació dels vèrtexs de la triangulació*

És necessari permetre que l'usuari pugui consultar les coordenades dels vèrtexs del terreny. Per tant, en seleccionar un vèrtex de la triangulació s'hauran de mostrar els valors corresponents a les coordenades X, Y i Z d'aquest.

- *Ajudes en la visualització de terrenys*

A part de poder visualitzar el terreny, ens interessarà disposar d'altres funcionalitats que ens ajudin a observar-lo amb més detall i d'una forma més interactiva i dinàmica. Les principals funcionalitats a l'hora de visualitzar el terreny són:

→ *Rotació de la càmera*

L'aplicació ha de poder visualitzar el terreny des de diferents punts de vista, fet que implica rotar el model en totes les direccions desitjades.

→ *Zoom*

L'aplicació ha de tenir la possibilitat de realitzar apropaments i allunyaments del terreny o d'una zona del terreny per tal de poder-lo visualitzar amb més o menys detall.

→ *Opcions de visualització del terreny*

El terreny ha de poder ésser visualitzat de diferents maneres/formes en funció del diseg de l'usuari. Per aquest motiu, l'aplicació ens ha de permetre visualitzar un terreny amb/sense vèrtexs, arestes i color.

- Emmagatzemar terrenys

L'aplicació ens ha de permetre poder desar els terrenys. Inicialment es requereix emmagatzemar tant la triangulació com el *PSLG*, però això dependrà del tipus de format escollit. En el *Capítol 5* de la memòria hi ha un apartat on comentarem amb més profunditat els tipus de formats utilitzats per tal de poder emmagatzemar el terreny.

- Desar imatges del terreny

Aquesta funcionalitat ens ha de permetre capturar i guardar imatges del contingut visible de l'Editor i del Visor en diferents formats gràfics, per tal de poder-les visualitzar sempre que vulguem i estudiar-les amb profunditat.

- Modificar configuració

Aquesta funcionalitat ens permetrà modificar característiques del programa com poden ésser el color de fons del visor, el color de l'editor, la mida de l'espai que s'utilitzarà per tal de mostrar el terreny, ...

Així doncs, aquests últims requeriments que acabem de comentar són requeriments que ja s'havien resolt en treballs anteriors, però que els hem utilitzat, adequat i/o modificat per tal d'incorporar-los a la nostra interfície i que d'aquesta manera ens ajudessin a complir els nostres objectius.

3.1.1 Actors del sistema

En aquest apartat durem a terme la identificació dels diferents **actors** que interactuaran amb la nostra aplicació. Recordem que un actor és una entitat externa (persona, dispositiu, procés, subsistema, ...) que interactua amb el sistema interpretant un determinat rol. Cal tenir present que els actors no són part del sistema que es construeix, així com entren informació al sistema i reben informació del sistema. Per tal de poder identificar-los, cal que ens fem la pregunta: **“Qui utilitzarà la nostra aplicació?”**.

En el nostre cas, veiem que només disposarem d'un sol actor, que serà **l'usuari**, el qual disposarà dels màxims privilegis i serà qui interactuarà en tot moment amb el sistema podent realitzar totes les accions que desitgi.

En la següent *Figura 3,1* podem observar els actors (en aquest cas, l'actor usuari) que interactuen amb el nostre sistema.

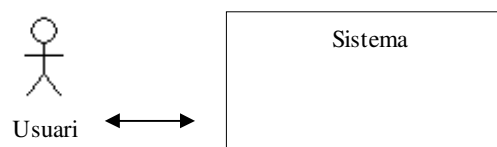


Figura 3.1 : Actors que interactuen amb el Sistema

3.1.2 Diagrames de casos d'ús

Els **casos d'ús** descriuen el comportament (funcionalitats) d'un sistema des del punt de vista de l'usuari. Així doncs, un cas d'ús representa un conjunt d'interaccions entre un o més actors i el sistema. Bàsicament, els casos d'ús ens representen els requeriments funcionals d'un sistema, ens descriuen què fa un sistema (no com ho fa) i en definitiva, dirigeixen tot el procés de desenvolupament d'un sistema.

Podem dir que els casos d'ús són descripcions de les funcionalitats del sistema independentment de la implementació. Estan basats en el llenguatge natural, de manera que puguin ésser accessibles per a tots els usuaris. Aquest tipus de diagrames han d'ésser el màxim de simples i entenedors possible ja que ens trobem en una primera fase dins l'estudi del programari (la fase d'anàlisi de requeriments).

Per tal de poder descriure a grans trets els requeriments funcionals del sistema de forma gràfica, així com les relacions existents entre els actors i cadascun dels casos d'ús que sorgeixen dels requisits esmentats anteriorment, es mostra el següent **diagrama de cas d'ús de context** (el diagrama general), que posteriorment anirem refinant, tot mostrant de forma ampliada i més precisa els diagrames d'aquelles funcionalitats més complexes, per tal de donar més claredat al lector.

Nota: Només realitzarem el diagrama de cas d'ús i les corresponents fitxes de les parts (funcionalitats) que hem afegit nosaltres a l'hora de realitzar el projecte. No es posaran els diagrames ni fitxes corresponents als requeriments que ja disposava el sistema a l'hora d'iniciar el projecte (que també els hem comentat anteriorment)

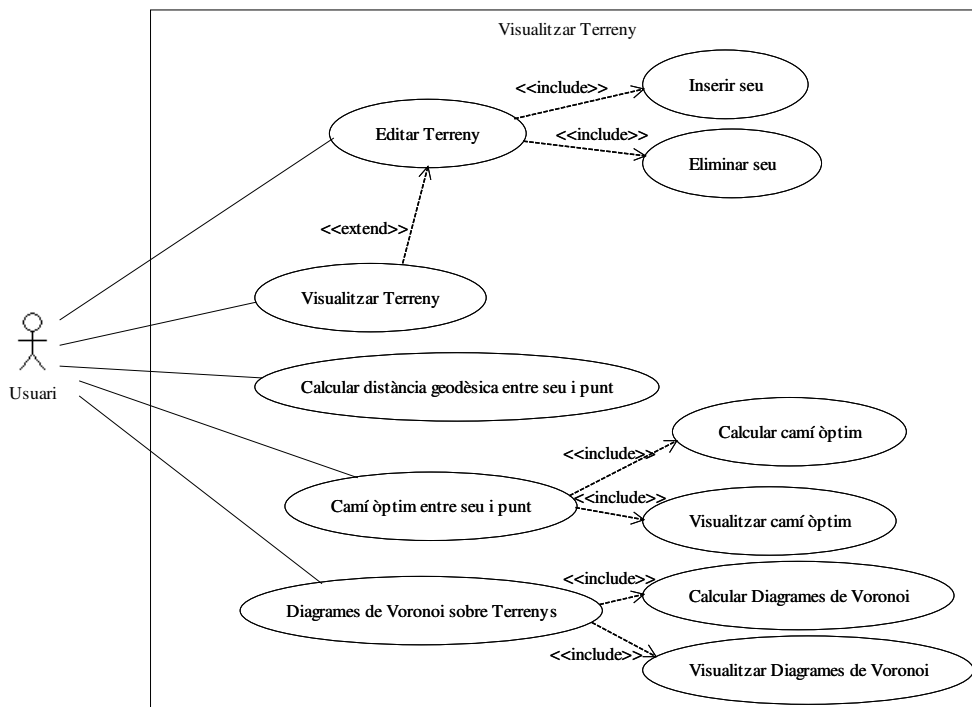


Figura 3.2 : Diagrama de cas d'ús de context del Sistema

Els casos d'ús amb funcionalitats més complexes són els que a continuació hem refinat per tal de poder-los observar amb més detall. En les figures següents els podrem anar trobant. Concretament, hem refinat els casos d'ús **Camí Òptim entre seu i punt** (Figura 3.3), **Diagrames de Voronoi sobre Terrenys** (Figura 3.4), **Editar Terreny** (Figura 3.5) i **Visualitzar Terreny** (Figura 3.6).

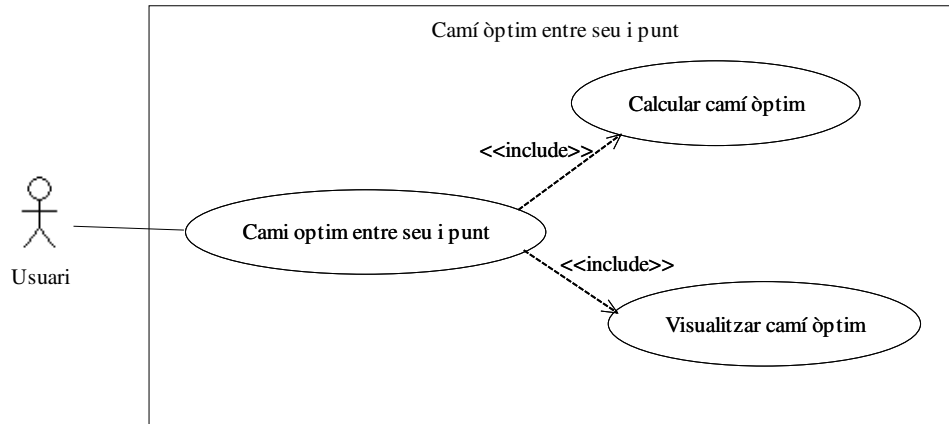


Figura 3.3 : Refinament del cas d'ús *Camí òptim entre seu i punt*, on tindrem les opcions de calcular el camí òptim i visualitzar aquest camí òptim. Tal i com veurem en capítols posteriors, aquest requeriment englobarà tota la part del càlcul de girs de desplegament, creació, propagació i intersecció de finestres, tècnica de backtracing, ...

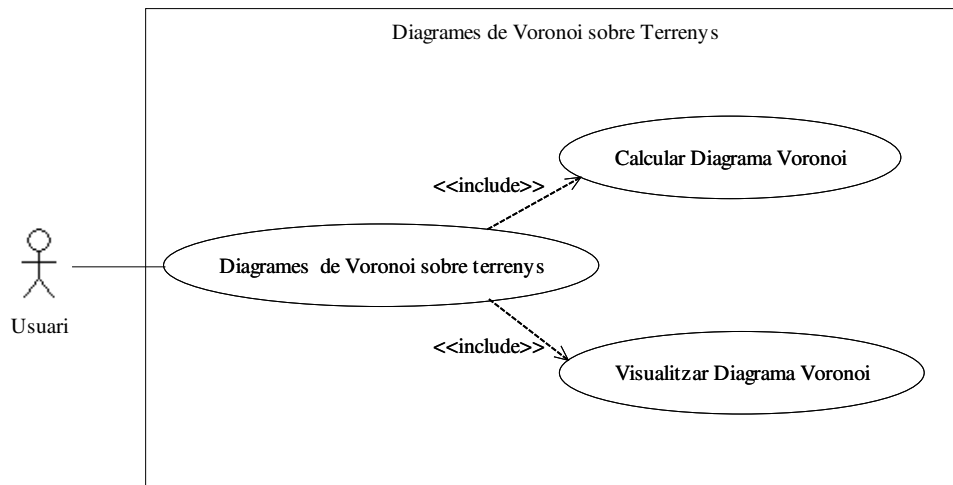


Figura 3.4 : Refinament del cas d'ús *Diagrames de Voronoi sobre Terrenys*, on tindrem les opcions de *Calcular el Diagrama de Voronoi* i *visualitzar aquest Diagrama de Voronoi*.

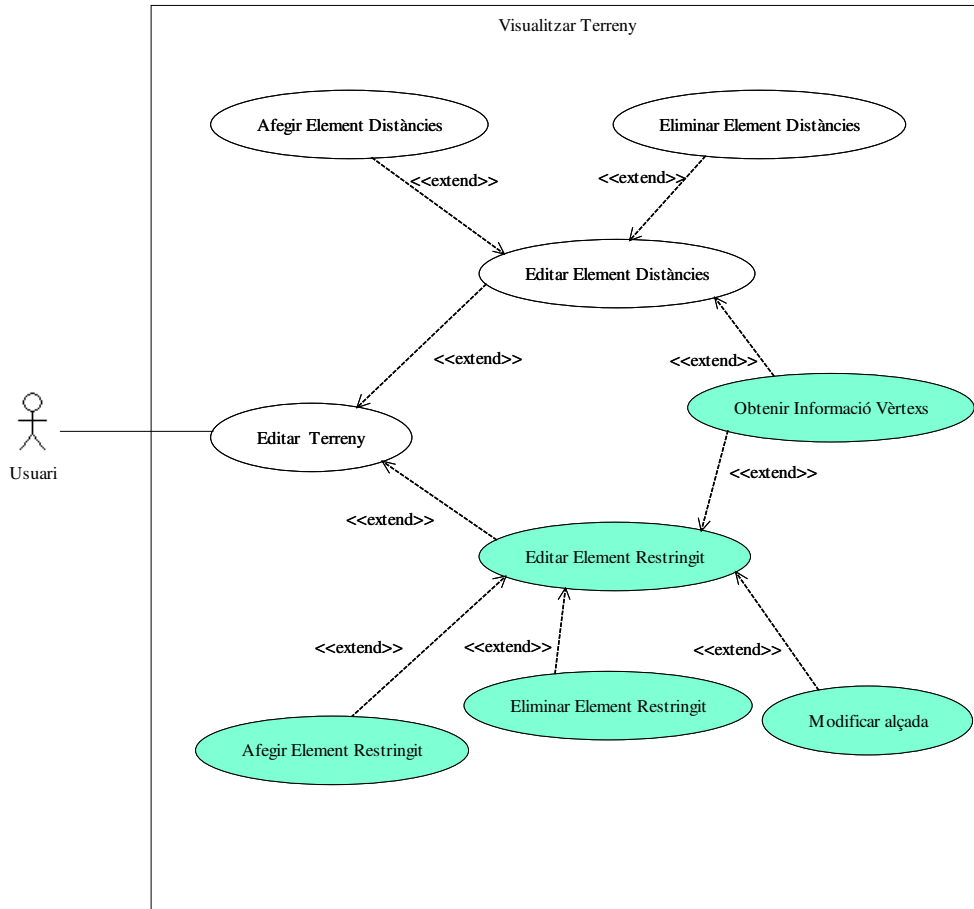


Figura 3.5 : Refinament del cas d'ús *Editar Terreny*, on durem a terme opcions diferents depenent de si estem tractant amb Elements de Distàncies (seus) o elements Restringits. La nostra part fa referència principalment a la inserció i eliminació d'elements de distàncies (seus). També hi haurà la possibilitat d'inserir punts de camí mínim.

Nota: En color les parts que gairebé no s'han modificat però que també intervenen en el diagrama

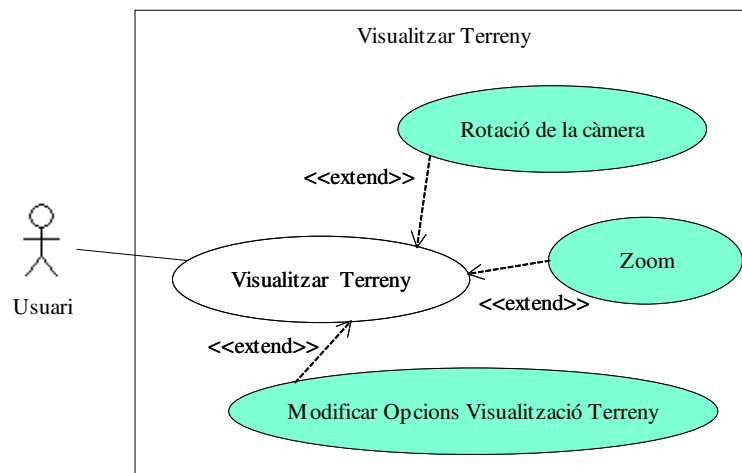


Figura 3.6 : Refinament del cas d'ús *Visualitzar Terreny*, on disposarem de diferents opcions de visualització, com poden ésser la rotació de la càmera, la modificació del zoom i la modificació d'altres opcions de visualització del Terreny.

3.1.3 Fitxes de casos d'ús

Per tal de conèixer amb més exactitud la funcionalitat dels requeriments funcionals, acompanyarem els **diagrames de casos d'ús** amb les seves corresponents **fitxes de casos d'ús** per tal d'aclarir encara més què és el que representa cada cosa.

Tal i com hem pogut observar, cada requeriment formarà un cas d'ús, ja que descriuen el comportament o funcionalitats del sistema quan aquest interactua amb un usuari extern o actor. Així doncs, cada cas d'ús s'especifica per mitjà d'una fitxa en la que es descriuen l'escenari principal i, si cal, els escenaris alternatius que tindrà. A continuació anem a observar les diferents fitxes de casos d'ús que tindrà el nostre sistema:

CAS D'ÚS	Afegir Element Distàncies (seu) al Terreny
Versió	1.0
Autors	Eduard Oliver
Descripció	L'usuari afegeix un element de distàncies (seu) en el terreny.
Actors	Usuari
Precondició	Disposar d'un terreny triangulat
Flux principal	1- Escollir la opció de treballar amb la modalitat distàncies 2- Escollir la seu (punt) que es vol afegir al terreny, i afegir-la a la posició indicada mitjançant el punter del ratolí.
Postcondició	El nou element inserit al terreny es visualitza immediatament en l'Editor 2D i també es visualitza en el Visor 3D en cas que tinguem la opció de <i>veure elements visió</i> activada.
Comentaris	Actualment només hi ha una versió operativa de la versió d'afegir punts (com a seus) en el terreny. Pròximament es farà operatiu el fet que es puguin afegir segments, polígons o poligonals que actuïn com a seus.

CAS D'ÚS	Eliminar Element Distàncies (seu) del Terreny
Versió	1.0
Autors	Eduard Oliver
Descripció	L'usuari elimina un element de distàncies (seu) del terreny.
Actors	Usuari
Precondició	Disposar d'un terreny triangulat i d'algun element (seu) inserit anteriorment per tal de poder-lo eliminar. Estar treballant amb la modalitat distàncies
Flux principal	1- Seleccionar l'element que es vol eliminar 2- Escollir la opció d'eliminar l'element (botó dret del ratolí)
Postcondició	L'element s'eliminarà del terreny. Aquest canvi es visualitzarà immediatament en l'Editor 2D i també es visualitzarà en el Visor 3D en cas que tinguem la opció de <i>veure elements visió</i> activada.
Comentaris	Actualment només hi ha una versió operativa de la versió d'eliminar punts (com a seus) en el terreny. Pròximament es farà operatiu el fet que es puguin eliminar també segments, polígons o poligonals que actuïn com a seus. En el cas d'aquests 3 últims, hi haurà la possibilitat d'eliminar tot l'element o eliminar-ne un tall.

CAS D'ÚS	Visualitzar terreny
Versió	1.0
Autors	Eduard Oliver
Descripció	L'usuari visualitza (tant per l'Editor 2D com per el Visor 3D) el terreny.
Actors	Sistema
Precondició	Haver creat un nou terreny, ja sigui generat de forma aleatòria, creat per l'usuari o obert des d'un fitxer anteriorment guardat.
Flux principal	1- Visualitzar el terreny.
Postcondició	El terreny es visualitzarà per la pantalla de l'usuari, tant en l'Editor 2D com en el Visor 3D.
Comentaris	Per defecte se'ns visualitzarà el terreny amb les arestes i els vèrtexs de la triangulació. Si l'usuari ho desitja, podrà visualitzar-lo sense alguna d'aquestes coses, així com canviar algunes altres opcions de visualització. (colors, textures, ...)

CAS D'ÚS	Calcular distància geodèsica entre seu i punt
Versió	1.0
Autors	Eduard Oliver
Descripció	El sistema ens calcula la distància geodèsica entre una seu i un punt del terreny, introduïts per l'usuari.
Actors	Sistema, Usuari
Precondició	Estar treballant amb la modalitat distàncies.
Flux principal	1- Inserir un element de distàncies (seu) en el terreny. 2- Realitzar tota la propagació de finestres a través del terreny 3- Inserir un punt qualsevol de camí mínim en el terreny 4- Escollir la opció de calcular la distància geodèsica entre una seu i un punt.
Postcondició	El sistema ens indicarà la distància geodèsica que hi ha entre la seu i el punt.
Comentaris	En el cas que disposem d'un conjunt de seus i d'un conjunt de punts, per a cada punt ens calcularà la distància geodèsica a la seu més propera. Actualment només hi ha una versió operativa de la versió de calcular la distància geodèsica entre una seu i un punt. Pròximament es farà operatiu el fet que tant la seu com el punt puguin ésser també segments, polígons o poligonals.

CAS D'ÚS	Calcular camí òptim entre seu i punt
Versió	1.0
Autors	Eduard Oliver
Descripció	El sistema ens calcula el camí òptim entre una seu i un punt del terreny, introduïts per l'usuari.
Actors	Sistema, Usuari
Precondició	Estar treballant amb la modalitat distàncies.
Flux principal	1- Inserir un element de distàncies (seu) en el terreny. 2- Realitzar tota la propagació de finestres a través del terreny 3- Inserir un punt de camí mínim qualsevol en el terreny 4- Escollir la opció de calcular el camí més curt entre una

	seu i un punt.
Postcondició	El sistema ens indicarà el camí més curt entre la seu i el punt.
Comentaris	En el cas que disposem d'un conjunt de seus i d'un conjunt de punts, per cada punt ens calcularà el camí més curt a la seu més propera. Actualment només hi ha una versió operativa de la versió de calcular el camí més curt entre seus (punts) i punts. Pròximament es farà operatiu el fet que tant la seu com el punt puguin ésser també segments, polígons o poligonals.

CAS D'ÚS	Visualitzar camí òptim entre seu i punt
Versió	1.0
Autors	Eduard Oliver
Descripció	El sistema ens permet visualitzar tant en l'Editor 2D com en el Visor 3D el camí òptim entre una seu i un punt del terreny, introduïts per l'usuari.
Actors	Sistema, Usuari
Precondició	Estar treballant amb la modalitat distàncies.
Flux principal	1- Inserir un element de distàncies (seu) en el terreny. 2- Realitzar tota la propagació de finestres a través del terreny 3- Inserir un punt de camí mínim qualsevol en el terreny 4- Escollir la opció de calcular el camí més curt entre una seu i un punt. 5- Visualitzar tant en l'Editor 2D com en el Visor 3D el camí més curt entre la seu i el punt
Postcondició	El sistema ens permet visualitzar el camí més curt entre una seu i un punt tant en l'Editor 2D com en el Visor 3D. Per defecte ja el visualitzarem immediatament en l'Editor 2D, en canvi, per visualitzar-lo per el Visor 3D haurem de tenir activada la opció <i>veure Elements Visió</i> .
Comentaris	En el cas que disposem d'un conjunt de seus i d'un conjunt de punts, per a cada punt ens calcularà i posteriorment es podrà visualitzar el camí més curt a la seu més propera. Actualment només hi ha una versió operativa de la versió de calcular i visualitzar el camí més curt entre seus (punts) i punts. Pròximament es farà operatiu el fet que tant la seu com el punt puguin ésser també segments, polígons o poligonals, i es puguin visualitzar també tant en l'editor 2D com en el Visor 3D.

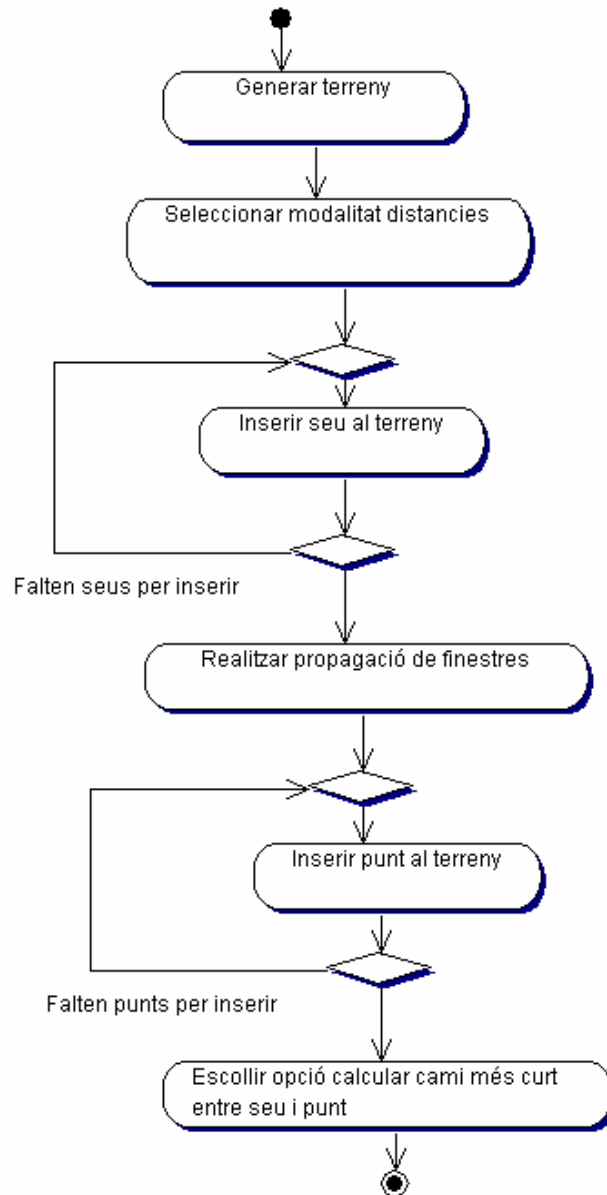
CAS D'ÚS	Calcular i Visualitzar Diagrama de Voronoi sobre Terreny
Versió	1.0
Autors	Eduard Oliver
Descripció	El sistema ens calcularà i ens permetrà visualitzar en l'Editor 2D el Diagrama de Voronoi sobre un terreny per a un conjunt de seus.
Actors	Sistema, Usuari
Precondició	Estar treballant amb la modalitat distàncies.
Flux principal	1- Inserir diverses seus (elements de distàncies) en el terreny. 2- Realitzar tota la propagació de finestres a través del terreny 3- Escollir la opció <i>veure Diagrama Voronoi</i> en l'Editor 2D.

Postcondició	El sistema ens calcularà i ens mostrarà en l'Editor 2D el Diagrama de Voronoi per al conjunt de seus que haguem inserit prèviament.
Comentaris	Actualment només hi ha una versió operativa de la versió de calcular i visualitzar els Diagrames de Voronoi, que funciona per seus que siguin punts. Pròximament es farà operatiu el fet que les seus puguin ésser també segments, polígons o poligonals.

3.1.4 Diagrames d'activitat

Els **diagrames d'activitat** són un dels tipus de diagrames UML que s'utilitzen per a realitzar el modelat dels aspectes dinàmics dels sistemes. Els diagrames d'activitat es centren en el flux d'activitats involucrades en un procés, generalment dins del marc d'un o diversos casos d'ús. Un diagrama d'activitat mostra com els processos que s'executen sobre diversos objectes depenen els uns dels altres. Aquests diagrames no proporcionen informació del comportament d'un objecte o de les col·laboracions entre objectes.

Normalment els diagrames d'activitat tenen la mateixa utilitat que les fitxes de casos d'ús, i es solen utilitzar de manera substitutiva o a vegades complementària. A continuació observarem els diagrames d'activitat d'algunes de les operacions més complexes i importants, per tal de reforçar la fitxa de cas d'ús i deixar del tot clar quin és el flux d'informació de cadascuna d'elles. Veurem el diagrama d'activitat de *"Calcular camí òptim entre seu i punt"* i *"Visualitzar Diagrama de Voronoi sobre Terreny"*

Calcular camí òptim entre seu i punt**Figura 3.7 :** Diagrama d'activitat del cas d'ús *Calcular camí òptim entre seu i punt*

Com es pot observar en el diagrama d'activitat, el primer que ens caldrà fer serà generar un terreny (de forma aleatòria, carregant-lo d'un fitxer ja existent o que l'usuari el construeixi manualment). Posteriorment, seleccionarem que treballarem amb la modalitat de treball de distàncies, i inserirem seus (punts) al terreny. Posteriorment, realitzarem la creació i posterior propagació de les finestres al llarg del terreny. Un cop ja haguem propagat totes les finestres, inserirem punts de camí mínim al terreny, que seran els punts a partir dels quals es calcularà la distància mínima (camí més curt) a la seu que tinguin més propera. Aquest resultat és el que es mostrarà i es visualitzarà en l'aplicació.

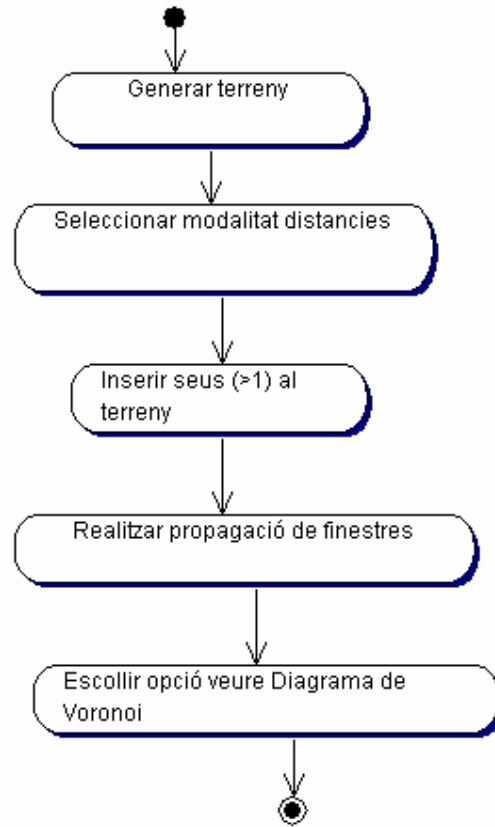
Visualitzar Diagrama de Voronoi sobre Terreny

Figura 3.8 : Diagrama d'activitat del cas d'ús *Visualitzar Diagrama de Voronoi sobre terreny*

Com es pot observar en aquest diagrama d'activitat, el primer que ens caldrà fer serà generar un terreny (de forma aleatòria, carregant-lo d'un fitxer ja existent o que l'usuari el construeixi manualment). Posteriorment, seleccionarem que treballarem amb la modalitat distàncies, i inserirem seus (punts) al terreny (per a l'opció de Diagrames de Voronoi haurem d'inserir més d'una seu). Posteriorment, realitzarem la creació i posterior propagació de les finestres al llarg del terreny. Finalment escollirem la opció de veure el Diagrama de Voronoi i immediatament visualitzarem el Diagrama de Voronoi per a aquell conjunts de seus en la nostra aplicació.

3.2 Requeriments No Funcionals

Un cop hem vist els requeriments funcionals, així com els diagrames de casos d'ús, les seves corresponents fitxes de casos d'ús i els seus corresponents diagrames d'activitat, anem a veure quins seran els **requeriments no funcionals** de la nostra aplicació, que com hem comentat són uns requeriments que descriuen restriccions, normalment quantificables, que venen imposades pel client o pel mateix problema.

Referent a la nostra aplicació, comentar que des del punt de vista de **seguretat** no es requereix cap tipus de control d'accés al programari, ja que es tracta d'un sistema *monousuari*, el qual només pot ésser utilitzat per un usuari cada vegada en una mateix PC. A més a més, no es disposa de dades confidencials ni d'alt risc, que puguin ésser falsejades, de manera que no tindria sentit un sistema de protecció de dades.

D'altra banda, comentar que l'aplicació ha estat implementada sobre una **plataforma Linux**. Alguns dels motius per els quals s'ha escollit aquesta plataforma a l'hora de desenvolupar l'aplicació són els següents:

- Linux és una plataforma que ofereix, en determinades ocasions, major fiabilitat i estabilitat
- Gran part de les eines utilitzades a l'hora de desenvolupar el projecte són de codi obert, amb llicència *GPL (GNU Public License)*, i per tant, és gratuït, de manera que no comporta cap cost addicional sempre i quan no es facin servir per a usos comercials.
- Altres avantatges són: flexibilitat de configuració, suport de diversos tipus de hardware, interoperativitat amb altres sistemes,...
- El fet que durant tota la carrera tècnica i la superior en una gran majoria d'assignatures haguem realitzat les pràctiques sobre aquesta plataforma també ha estat un motiu per el qual s'ha escollit

La implementació de l'aplicació s'ha dut a terme bàsicament amb el SuSE Linux 9.1, i les proves s'han realitzat principalment des de dos equips diferents:

- Processador: Intel Pentium IV 2.66 GHz – 512 MB de memòria RAM
 - o Targeta Gràfica: NVIDIA GeForce 4 MX 440 64MB
- Processador: Intel Pentium IV 3.00 GHz – 1 GB de memòria RAM
 - o Targeta Gràfica: NVIDIA GeForce 6600 256 MB

Comentar també el fet que per tal de poder executar i visualitzar correctament l'aplicació cal disposar d'una **targeta gràfica** que tingui força memòria (com més memòria tingui amb més fluïdesa i qualitat podran ésser visualitzats els terrenys) i que permeti acceleració 3D, sinó podríem tenir problemes per visualitzar principalment la part referent als *Diagrames de Voronoi*. (es recomana tenir com a mínim una targeta gràfica de 256 MB, ja que en el primer dels 2 equips que he apuntat aquesta part no es podia visualitzar correctament). A part de la targeta gràfica, també es recomana disposar d'espai de disc i d'uns 50 MB de memòria RAM lliures, tot i que avui en dia aquests problemes no són tant importats ja que aconseguir augmentar la capacitat d'un ordinador és econòmicament assequible.

Tant el **rendiment** de l'aplicació com el **temps de resposta** depenen lògicament del tipus de memòria de la qual disposi l'ordinador a utilitzar. En funció de la mida del terreny, és a dir, del nombre de triangles que aquest contingui, aquests paràmetres es podran veure afectats ja que, com més gran sigui el model, més memòria serà requerida.

Pel que fa a la **resolució** de la pantalla, per tal de poder visualitzar-lo correctament es recomana una resolució de 1024 x 768. Evidentment, com més gran sigui la resolució de la pantalla millor qualitat visual obtindrem.

Un altre punt a destacar són els **perifèrics o dispositius d'entrada** amb els quals es realitzarà la interacció amb l'aplicació. L'usuari podrà interactuar amb l'aplicació bàsicament a través del ratolí, ja que la interfície consta de botons i menús desplegable. D'altra banda, des del teclat es podran complementar algunes de les funcionalitats del programari, tant de l'Editor com del Visor, ja sigui introduint paràmetres d'entrada (principalment quan vulguem utilitzar una demo o realitzar algun testeig d'alguna operació/funció concreta) o visualitzant missatges informatius.

Un altre requeriment no funcional per tal de poder treballar correctament amb el programa és tenir instal·lades en el sistema les **llibries gràfiques** de les **Qt (versió 3.3.3)** i **OpenGL (versió 1.4)**. Tot i això, com que les llibries escollides són multiplataforma, fent relativament pocs canvis sobre l'aplicació aquesta ja podria funcionar correctament en qualsevol altra plataforma.

La interfície gràfica d'usuari va ésser construïda a partir de les **Qt**, ja que permet treballar còmodament amb **OpenGL** i, per tant, totes les modificacions realitzades sobre aquesta han hagut de respectar aquestes llibries. Normalment, qualsevol sistema Linux que disposi de l'entorn d'escriptori KDE disposa d'una versió de les **Qt** compilada i amb les eines necessàries per tal de desenvolupar programari. En cas que no se'n disposi, serà relativament senzill que l'usuari es pugui instal·lar les llibries descarregant-les d'Internet. Pel que fa a les biblioteques d'**OpenGL**, cal dir que aquestes depenen de la targeta gràfica, ja que són contingudes pels drivers d'aquesta. En el cas que la targeta gràfica no disposi dels drivers necessaris, caldrà instal·lar algunes biblioteques alternatives, com podrien ésser per exemple les **Mesa**, les quals permeten executar aplicacions via software, amb l'inconvenient que el rendiment disminueix. Tal i com ja hem comentat

anteriorment, una de les característiques que ha de tenir la targeta gràfica per tal de poder executar l'aplicació sense problemes és que disposi d'acceleració 3D.

Per tal de **compilar** l'aplicació, només ens caldrà executar el *makefile* propi d'aquesta. Aquest ha estat dissenyat a través de l'eina **KDevelop**, la qual ens permet editar, compilar i arrencar el programari.

Finalment, comentar que els **objectius de disseny o requeriments de qualitat** són força amplis ja que es tracta d'un sistema bastant fàcil d'utilitzar i de mantenir. Tot i això, s'ha intentat que el programari sigui fiable, estable i robust, intentant obtenir sempre el mínim temps de resposta possible. Pel que fa a la robustesa de l'aplicació, s'ha realitzat un control d'errors de diversos àmbits, així com s'ha dut a terme un extens joc de proves per tal de testejar cadascuna de les funcions, des de les més bàsiques a les més complexes. Finalment, i des del punt de vista evolutiu, comentar que l'aplicació pot ésser ampliada posteriorment amb noves funcionalitats, les quals explicarem més detalladament en el *Capítol 7*, en el qual parlarem de possibles millores i treball futur.

Capítol 4

Anàlisi del Sistema

En aquest capítol de la memòria realitzarem bàsicament l'**anàlisi del sistema**, posant especial èmfasi en l'estructura de classes dissenyada per tal de implementar el projecte. Un cop definits els requeriments principals del sistema (*Capítol 3* de la memòria), el principal objectiu d'aquesta nova etapa d'anàlisi serà obtenir una comprensió precisa de les necessitats del sistema, és a dir, s'encarregarà de la investigació del problema a resoldre (*què*), sense ocupar-se de trobar-ne una solució (*com*). Durant aquest procés es realitzarà una mena de traducció dels requeriments comentats en el capítol anterior a un llenguatge més formal segons *l'Enginyeria del Programari*. En definitiva, el propòsit de l'anàlisi orientat a objectes consisteix en definir l'anomenat *model del domini*, és a dir, totes les classes que seran necessàries per a resoldre el problema, així com els atributs i les operacions (mètodes) associades a cadascuna de les classes.

Utilitzarem diferents diagrames UML per ajudar-nos a il·lustrar el comportament del sistema, com poden ésser el **diagrama de classes** i **diagrames de seqüència**. Comentar que en aquesta etapa d'anàlisi ja presentarem els diagrames en la seva etapa final, és a dir, incorporant decisions de disseny. Aquest fet es dona perquè la orientació a objectes permet que els mateixos models siguin utilitzats de forma iterativa en les diferents etapes del software, fent-los créixer des dels requeriments fins a la implementació.

4.1 Diagrama de classes

Podríem dir que els **diagrames de classes** són els més utilitzats en el modelat de sistemes orientats a objectes. Un diagrama de classes s'utilitza per tal d'analitzar, o bé especificar, els requeriments a l'hora de construir el model d'anàlisi. Bàsicament conté un conjunt de *classes* i les *relacions* que s'estableixen entre aquestes classes.

Els diagrames de classes ens serviran, per tant, per explorar conceptes del domini del problema a la fase de definició de requeriments, analitzar o especificar requeriments a l'hora de construir el model d'anàlisi i descriure detalladament el programari a construir a la fase de disseny. Així doncs, un diagrama de classes ens proporciona una visió estàtica del sistema a desenvolupar, ja que mostra les classes que interactuen sense mostrar què passa quan aquestes ho fan.

En aquest apartat veurem quina és l'estructura de classes utilitzada així com els components de cadascuna de les classes del sistema. També veurem com estan relacionades entre si totes aquestes classes. Tot aquest conjunt de classes, juntament amb les interrelacions existents entre elles, permeten descriure totes les operacions definides en la fase d'anàlisi de requeriments.

Aquest programari, des del punt de vista de la interfície tindrà dues parts: *l'Editor 2D* i el *Visor 3D*. A través d'aquestes dues parts s'aniran repartint les funcionalitats de l'aplicació, tot i que serà en *l'Editor 2D* allà on realitzarem la majoria de funcionalitats. Les classes dedicades a la implementació de la interfície utilitzaran mètodes de les llibreries *d'OpenGL* i *Qt*.

En la *Figura 4.1* de la pàgina següent podem observar el diagrama complet de classes. Posteriorment les anirem agrupant per paquets (segons el color de fons col·locat en cada classe) i anirem explicant quin és el funcionament de cada classe dins de l'aplicació.

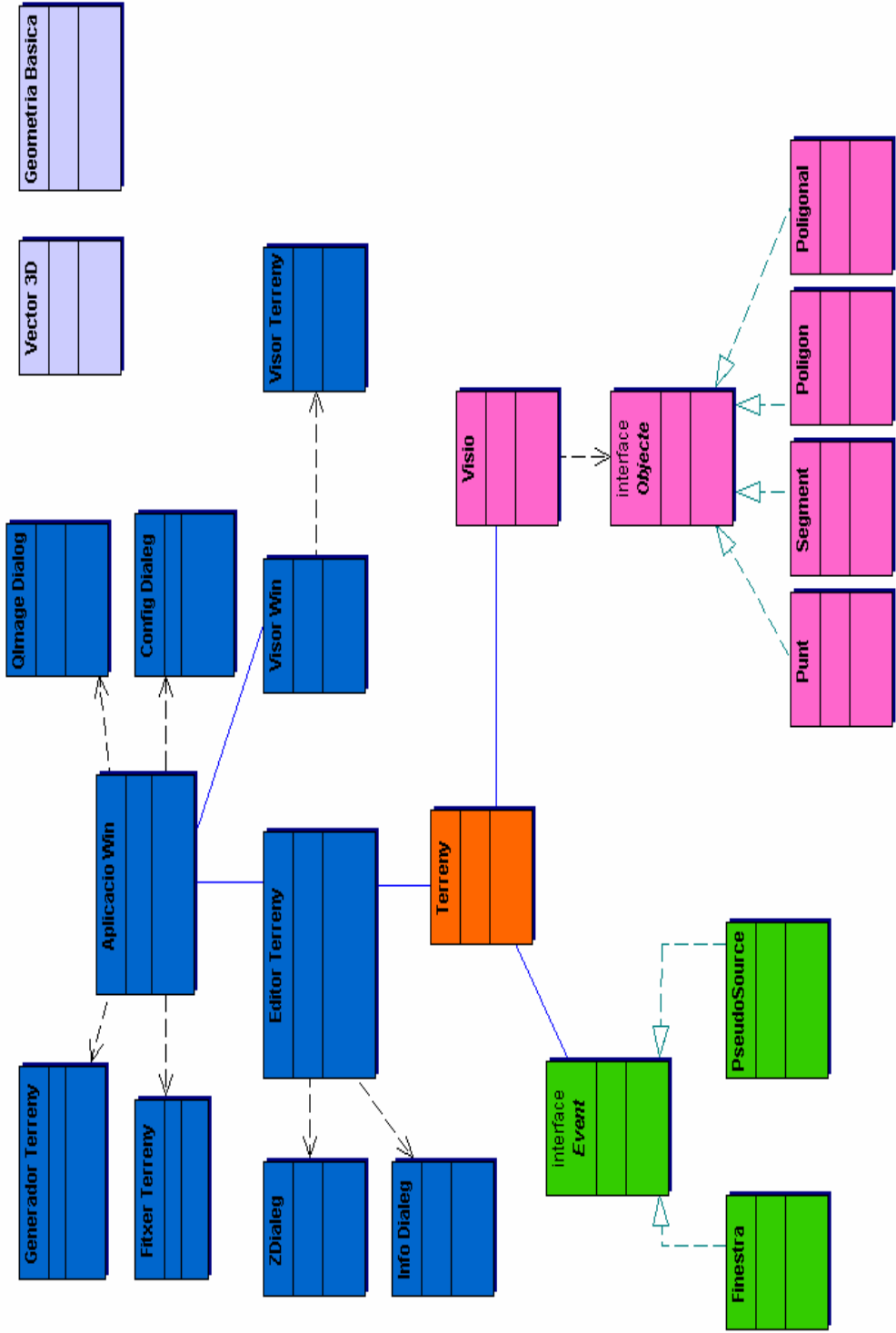


Figura 4.1 : Diagrama de classes

Tal i com s'ha pogut observar en la *Figura 4.1*, hem agrupat les classes en diversos paquets. A cada paquet li hem assignat un color diferent, segons a quina part del programari feia referència. Anem a veure quina ha estat la classificació que hem realitzat:

- **Paquet Interfície gràfica i programa principal** → Classes que fan referència al programa principal i a la interfície gràfica de l'aplicació, tant de l'Editor 2D com del Visor 3D.
- **Paquet Terreny** → Classes que fan referència al Terreny pròpiament dit, que inclouen també totes aquelles classes que ens permeten emmagatzemar tota la informació referent al terreny.
- **Paquet Elements Distàncies i Visió** → Classes que fan referència a elements de visió o distàncies que es poden introduir o eliminar en el terreny. Cada classe de les que deriven (Punt, Segment, Polígon i Poligonal) porta uns mètodes propis de la seva classe que permeten realitzar operacions pròpies dels objectes d'aquella classe.
- **Paquet Events** → Classes exclusivament creades per a realitzar tota la part de càlcul de camins òptims, que ens serviran per realitzar tota la creació i propagació de finestres i pseudosources al llarg del terreny.
- **Paquet Auxiliar** → Classes auxiliars que també s'han utilitzat per tal d'ajudar-nos a realitzar aquest projecte.

A continuació anem a descriure amb més detall i a veure el funcionament de cadascuna de les classes que conformen el diagrama de classes, seguint també la mateixa classificació per colors que acabem de comentar anteriorment.

Nota: *En el diagrama de classes de la Figura 4.1 no s'han posat totes les classes de les quals disposa el programa, sinó que només s'han tingut en compte les més importants. Les classes que no figuren en el diagrama és que no s'han modificat i gairebé no s'han utilitzat.*

4.1.1 Paquet Interfície gràfica i programa principal

→ Classe Aplicacio Win

La classe **Aplicació Win** és la interfície del programari. Conté les instàncies de tots els elements de la interfície tant per la finestra de l'Editor 2D com per la del Visor 3D. Permet realitzar la connexió entre les dues finestres.

A partir d'ella s'inicialitzen tots els elements gràfics de l'Editor 2D (finestres, menús, barres d'eines, ...) i ens permet la seva visualització. És l'encarregada d'emmagatzemar les opcions de configuració de tot el programari a través de la

variable *prefs*, a través de la qual podrem accedir a informació relacionada amb els elements del programari que l'usuari pot modificar.

A través d'aquesta classe es realitzen les crides a les diferents classes de l'aplicació que implementen les funcionalitats del sistema. Així doncs, podem dir que es tracta d'una classe de control que regula el flux d'operacions entre gran part de les classes de l'aplicació.

→ **Classe Editor Terreny**

Aquesta classe és una de les més importants de l'aplicació, ja que és l'encarregada de realitzar totes i cadascuna de les crides d'edició. Permetrà a l'usuari visualitzar el terreny a través de l'Editor 2D, modificar-lo i consultar i realitzar les operacions que desitgi. Podríem dir que l'**Editor Terreny** és l'àrea de dibuix on l'usuari pot realitzar cadascuna de les operacions anteriorment esmentades.

L'usuari podrà afegir, seleccionar, eliminar i modificar qualsevol element representat en el terreny, tenint en compte les diverses funcionalitats implementades. També en aquesta classes es desenvoluparan la majoria de mètodes per resoldre la problemàtica del projecte que ens envolta.

La classe *Editor Terreny* hereta directament de *QGLWidget*, classe pròpia de les Qt. Aquesta, ens proporciona les eines necessàries per a implementar una àrea de dibuix dins la qual poder dibuixar elements geomètrics de qualsevol mena a través de les comandes d'*OpenGL*. Cal dir que en la implementació d'aquesta classe es tenen en compte events necessaris per a poder controlar la interacció de l'usuari amb el programari i executar en cada moment les operacions que calguin.

A partir d'aquesta classe controlarem la triangulació del model representat mitjançant un conjunt de classes interrelacionades. Principalment s'enviaran missatges a la classe **Terreny**, la qual serà l'encarregada de traslladar-los a quin convingui. Serà l'encarregada de dur a terme les modificacions requerides per l'usuari sobre el terreny i mostrar la resposta, si cal, a través de missatges informatius.

Principalment, l'usuari disposarà de diversos modes de treball, segons els quals la mateixa acció realitzada per l'usuari provocarà diferents operacions en el terreny. Per exemple, si volem afegir un punt en mode de visibilitat, aquest punt serà considerat com un element restringit del terreny i es tornarà a triangular la malla, però si estem en mode distàncies, aquest punt es comportarà com una seu, i podrem començar ja a realitzar tota la propagació de finestres i càlcul del camí més curt.

Finalment comentar que existirà un event provocat que permeti comunicar l'*Editor 2D* amb el *Visor 3D* sempre que es produeixi qualsevol canvi en el terreny, per tal que aquest últim refresqui també la visualització. La funció encarregada de realitzar l'avís és l'anomenada *terrenyModificat()*.

→ **Classe Visor Win**

Es tracta de la classe que permet construir la finestra del *Visor 3D*. De la mateixa manera que la classe *Aplicacio Win*, aquesta disposa de les funcionalitats bàsiques que permeten construir els botons, menús, barra d'estats, ... necessaris. Per tant, implementa la interfície del Visor amb les funcions de la classe *Visor Terreny*.

→ **Classe Visor Terreny**

Aquesta classe permet representar l'àrea de dibuix corresponent al *Visor 3D*. Permet, per tant, visualitzar el terreny mitjançant diverses opcions. Per tal de poder representar el model, accedeix a la classe **Terreny**, la qual li proporciona la informació. De la mateixa manera que passava en la classe *Editor Terreny*, la classe **Visor Terreny** també hereta de la classe *QGLWidget*, per tant, podrà representar elements tridimensionals a través de les comandes d'*OpenGL*.

Les principals característiques del tractament de terrenys en 2.5D són:

- Moviment lliure de la càmera que pot ésser governat per l'usuari a través de diverses opcions. Gràcies als events programats, es podran dur a terme les rotacions i translacions que es desitgi.
- Zoom a través del desplaçament de la roda central (en cas que en tingui) del ratolí, o bé, dels botons de la finestra, permetent apropar o allunyar la càmera del terreny
- Visualització en tot moment del terreny, així com dels elements restringits introduïts, de les seues inserides i del camí òptim per anar d'un punt a una seu.
- Visualització de totes les finestres del terreny un cop propagades.

→ **Classe Generador Terreny**

A partir de la classe **Generador Terreny** podem crear terrenys pseudo-aleatoris. Aquesta classe engloba un conjunt d'algorismes que permeten realitzar el càlcul de punts aleatoris. Només és necessari indicar quina serà la mida màxima que volem que tingui el terreny, seleccionar el mètode de generació i escollir els paràmetres necessaris. Es tracta d'una eina útil durant el procés de proves, ja que permet generar terrenys força reals.

Aquesta classe doncs, ens permet crear terrenys de certa complexitat, que serien difícils de generar manualment. Aquesta classe serà cridada des de

la classe *Aplicacio Win* en el moment en què l'usuari vulgui crear un nou terreny aleatori.

→ **Classe Fitxer Terreny**

Es tracta de la classe que permet gestionar el tractament amb els fitxers que contenen terrenys visualitzables en l'Editor. Ens permetrà realitzar tota la part de carregar terrenys, emmagatzemar-los en diferents formats, ...

→ **Classe Config Dialog**

Es tracta d'una classe de configuració a través de la qual es pot obtenir tota aquella informació que l'usuari desitgi modificar. Implementa una finestra de diàleg on es mostren les parts de l'*Editor 2D* que poden ésser modificades, inicialment amb el valor actual. Principalment es poden modificar els colors referents al color de fons, color dels vèrtexs i color de les arestes.

→ **Classe QImage Dialog**

La finalitat d'aquesta classe és poder guardar el contingut de l'Editor i/o del Visor en un fitxer amb format gràfic. Consisteix en permetre fer una mena de fotografia del contingut visible per a cadascuna de les finestres en qualsevol instant de temps. Aquesta classe implementa una finestra de diàleg a partir de la qual l'usuari tant sols haurà d'indicar el nom i el format amb el qual vol desar la imatge.

→ **Classe ZDialog**

La classe *ZDialog* implementa una finestra de diàleg a través de la qual l'usuari haurà d'indicar un valor d'alçada per a un punt determinat. Aquesta finestra serà mostrada en diverses operacions, com per exemple, en el cas d'afegir un nou punt de restricció al terreny (recordem que per afegir una seu (element de distàncies) no s'haurà de cridar a aquesta classe). El valor d'alçada estarà limitat per les mides del terreny, que inclouen la coordenada Z.

→ **Classe Info Dialog**

Es tracta de la classe que descriu una finestra de diàleg amb la qual es mostrarà la informació sobre les coordenades d'un punt (vèrtex) seleccionat, ja sigui un punt restringit, una seu o un punt de camí mínim.

4.1.2 Paquet Terreny

→ **Classe Terreny**

Podríem dir que una de les classes principals és la classe **Terreny**. Tal i com es pot veure en el diagrama de classes, aquesta juga un paper molt important en l'aplicació ja que gestiona cadascuna de les operacions a dur a

terme amb la triangulació. Podríem dir que aquesta classe serveix de pont entra la interfície d'usuari i les funcionalitats del sistema..

Des d'aquesta classe podem accedir a la triangulació de *Delaunay* que té el terreny, i podem realitzar-hi moltes modificacions. De fet, la majoria de crides que fem des de la classe *Editor Terreny* aniran a parar a noves crides que són implementades a la classe *Terreny*, que serà la que realment modificarà l'estructura interna del terreny, afegint punts o seus, eliminant-los, canviant-los de posició, així com també ens permetrà accedir a la informació de cada vèrtex de la triangulació, entre d'altres coses. Així doncs, aquesta classe és imprescindible per tal de dur a terme els objectius del nostre projecte.

4.1.3 Paquet Elements Distàncies i Visió

→ Classe Visio

Es tracta de la classe que ens permet guardar i realitzar totes les operacions pertinents amb els objectes de visió que l'usuari hagi introduït sobre el terreny, ja siguin elements restringits o elements de distàncies (seus i punts de camí mínim).

→ Classe Objecte

Es tracta d'una classe abstracta que defineix els mètodes comuns per a les figures geomètriques que conté (Punt, Segment, Polígon i Poligonal).

→ Classe Punt

Classe que hereta de la classe *Objecte*. Com a atributs elementals disposa de les coordenades X, Y i Z del punt. Implementa els mètodes que permeten gestionar i tractar els punts del terreny, tant siguin elements restringits com elements de distàncies (seus i punts de camí mínim).

→ Classe Segment

Classe que hereta de la classe *Objecte*. Conté com a atributs principals els punts origen i destí que defineixen el segment. Implementa totes aquelles operacions necessàries per a la gestió de segments.

→ Classe Poligon

Classe que hereta de la classe *Objecte*. Conté com a atributs principals el vector de punts que defineixen la figura geomètrica i el nombre de costats que té la figura geomètrica. Implementa totes aquelles operacions necessàries per a la gestió de polígons.

→ **Classe Poligonal**

Classe que hereta de la classe *Objecte*. Conté una sèrie de punts units per segments. Implementa totes aquelles operacions necessàries per a la gestió de poligonals.

4.1.4 Paquet Events

→ **Classe Event**

Classe abstracta que conté tots els mètodes que s'implementen en les classes *Finestra* i *PseudoSource*. Cada event serà d'un tipus diferent (*Finestra* o *PseudoSource*) i tindrà un pes associat que ens servirà per tal d'ordenar-los en una cua de prioritats, que la utilitzarem per establir l'ordre de propagació dels events (*Finestres* i *PseudoSources*) al llarg del terreny.

→ **Classe Finestra**

Classe que deriva de la classe *Event*. Les **Finestres** seran un tipus d'event. Conté com a atributs principals l'estructura (tupla) definida per a gestionar les finestres explicada en el *Capítol 2*. Implementa totes aquelles operacions necessàries per a la gestió de finestres, que ens seran necessàries per tal de realitzar amb èxit els principals objectius del nostre projecte.

→ **Classe PseudoSource**

Classe que deriva de la classe *Event*. Els **PseudoSources** seran un tipus d'event. Conté com a atributs principals l'estructura (tupla) definida per a gestionar els pseudosources explicada en el *Capítol 2*. Implementa totes aquelles operacions necessàries per a la gestió de pseudosources, que recordem que seran seus puntuals, i que sempre seran vèrtexs de la triangulació del terreny.

4.1.5 Paquet Auxiliari

→ **Classe Geometria Bàsica**

Tal i com es pot observar en el diagrama de classes, la classe **Geometria Bàsica** no es troba associada a cap classe, ja que s'utilitza en diversos llocs de l'aplicació. Es tracta d'un fitxer de capçalera dins del qual s'hi realitzen les definicions globals necessàries per el sistema. Dins d'aquesta hi trobarem, per exemple, les estructures enumerables utilitzades per descriure el mode de treball, les estructures utilitzades per guardar informació sobre una cara o un vèrtex de la triangulació, la tolerància permesa a l'hora de realitzar els càlculs de distàncies, ...

→ Classe Vector3D

La classe **Vector3D** ens proporciona totes aquelles operacions necessàries per tal de poder treballar amb vectors tridimensionals. Aquesta classe és utilitzada en diversos llocs de l'aplicació a l'hora de realitzar determinats càlculs. Tenim la possibilitat de definir vectors de dues maneres diferents: amb dos punts o directament amb les coordenades X, Y i Z del vector.

Nota: *També s'han utilitzat, tot i que no les comentarem de forma exhaustiva perquè s'han tocat molt per sobre, les classes Pla, Línia i Triangle.*

4.2 Diagrames de seqüència

Els **diagrames de seqüència** són un tipus de diagrames d'interacció que detallen com s'executen les operacions en funció del temps: quins missatges són enviats, per quin objecte, a quin objecte i en quin moment. Així doncs, ens mostraran la interacció existent entre els objectes mitjançant l'enviament de missatges. Aquest tipus de diagrames s'organitzen en funció del temps i, per tant, permeten observar l'ordre amb el qual són enviats els missatges. Seran bastant útils per tal de modelar possibles escenaris.

A continuació, doncs, després d'haver especificat els casos d'ús (amb les seves corresponents fitxes i diagrames d'activitat) i el diagrama de classes del sistema, podem observar alguns dels diagrames de seqüència més complexos per tal de poder acabar de lligar els diagrames anteriors. A més a més, a partir d'aquests, podem entendre amb més claredat la seqüència de crides que es produeixen majoritàriament després de dur a terme algun event (efectuat per l'usuari en utilitzar algun dels casos d'ús definits).

4.2.1 Diagrama de seqüència Inserir seu al terreny

Una de les operacions bàsiques que l'aplicació ha de permetre realitzar és inserir una seu (element de distàncies) al terreny. Per aquest motiu, mostrarem a continuació el diagrama de seqüència que permet afegir un punt (seu) al terreny. (Recordem que de moment només es permeten inserir punts que actuïn com a seus, tot i que en el futur s'ampliarà a segments, polígons i poligonals).

Tal i com es pot observar en la *Figura 4.2*, l'usuari interactua amb el sistema a través de la classe *Aplicacio Win*. Aquest envia les peticions a la classe de control *Editor Terreny* encarregada de distribuir cadascuna de les operacions a cadascuna de les entitats amb la capacitat de resoldre-les.

Suposant que ja estem treballant amb un terreny que hem generat anteriorment, el primer que caldrà fer és que l'usuari esculli que està treballant

amb la modalitat distàncies. Això fa que es cridi un mètode de *Editor Terreny* que fa que es modifiqui el mode de treball a distàncies i es comença a realitzar la configuració del mode distàncies .

Posteriorment, l'usuari indicarà quin és l'element que vol inserir. En aquest cas, inserirem un punt que faci de seu. L'usuari seleccionarà que vol un punt, i immediatament es canviarà un altre mode de treball de *Render* a *GL_RENDER_PUNT* .

Tot seguit, l'usuari ja pot iniciar la inserció del punt simplement produint un event amb el clic del ratolí en un punt del terreny (defineix les coordenades X i Y del punt, ja que la coordenada Z no l'haurà d'introduir, sinó que el mateix programa ja ens la calcularà i ens ubicarà la seu ben bé a sobre el terreny). Aquesta operació farà que la classe *Editor Terreny* creï una instància de *Punt* amb els valors indicats per l'usuari. A continuació es cridarà la funció corresponent a la inserció de punt (seu) a través de la classe *Terreny*. Finalment, ja podrem visualitzar per pantalla la nova seu (punt) inserida. Aquest diagrama és gairebé igual tant per inserir punts que actuïn com a seu com punts de camí mínim inserits en el terreny per tal de calcular-ne la distància respecte la seu.

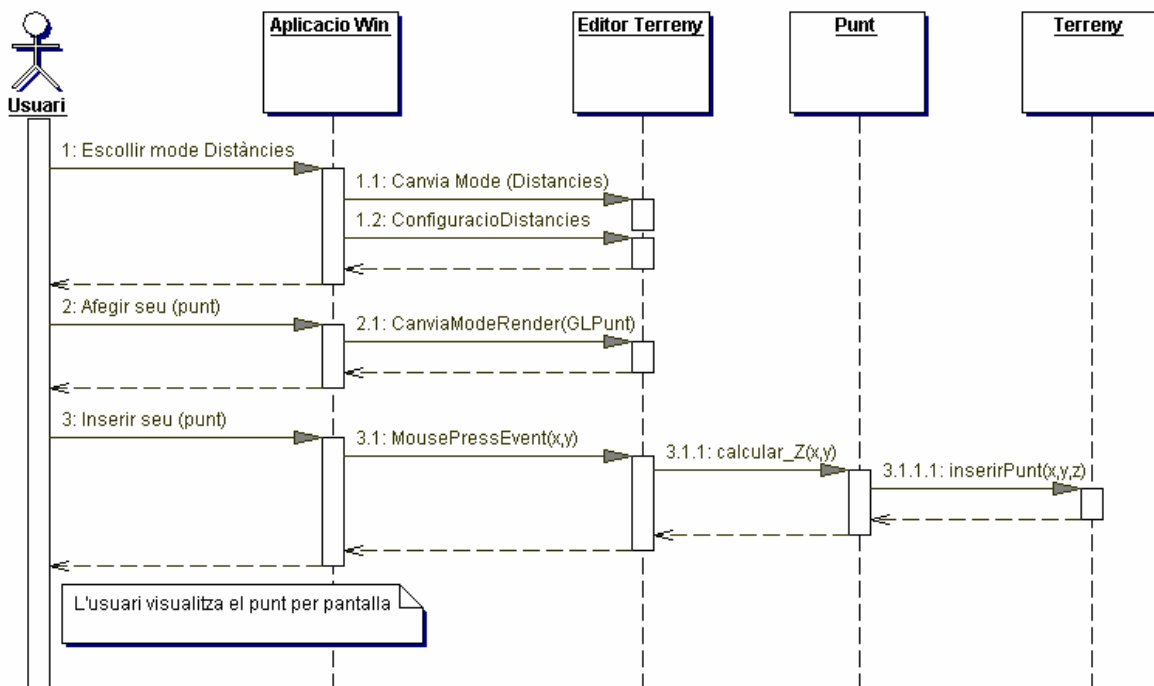


Figura 4.2 : Diagrama de seqüència *Inserir seu al Terreny*

4.2.2 Diagrama de seqüència Camí Òptim entre seu i punt

Una altra de les operacions més complexes i importants que ens ha de permetre realitzar la aplicació és la de calcular i visualitzar el camí òptim que hi ha entre una seu i un punt qualsevol del terreny. En cas que disposem de vàries seus i diversos punts, el mètode ens ha de retornar el camí mínim de cada punt a la seu més propera.

Tal i com es pot observar en la *Figura 4.3*, l'usuari interactua amb el sistema a través de la classe *Aplicacio Win*. Aquest envia les peticions a la classe de control *Editor Terreny*, que és l'encarregada de distribuir cadascuna de les operacions a cadascuna de les entitats amb la capacitat de resoldre-les.

Suposant que ja estem treballant amb un terreny que hem generat anteriorment, els primers passos que caldrà seguir són els mateixos que hem realitzar per inserir una seu (explicats en el diagrama de seqüència anterior).

Un cop ja tinguem la seu inserida en el terreny, premerem una opció que ens cridarà un mètode de la classe *Editor Terreny* que ens realitzarà, a grans trets, la inicialització de finestres i pseudosources, la posterior propagació de finestres i pseudosources (tenint en compte també si les finestres es solapen),... És a dir, ens anirà posant finestres al llarg de tot el terreny.

Posteriorment, inserirem un punt de camí mínim (que no actuï com a seu) en el terreny, que serà el punt sobre el qual calcularem el camí més curt a la seu. En aquest punt també només li indicarem les coordenades X i Y amb el ratolí, la coordenada Z ja ens la calcularà el sistema i ens la posarà sobre el terreny. Finalment cridarem al mètode *camiMesCurt()* que serà el que ens retornarà i ens permetrà visualitzar el resultat final per la pantalla.

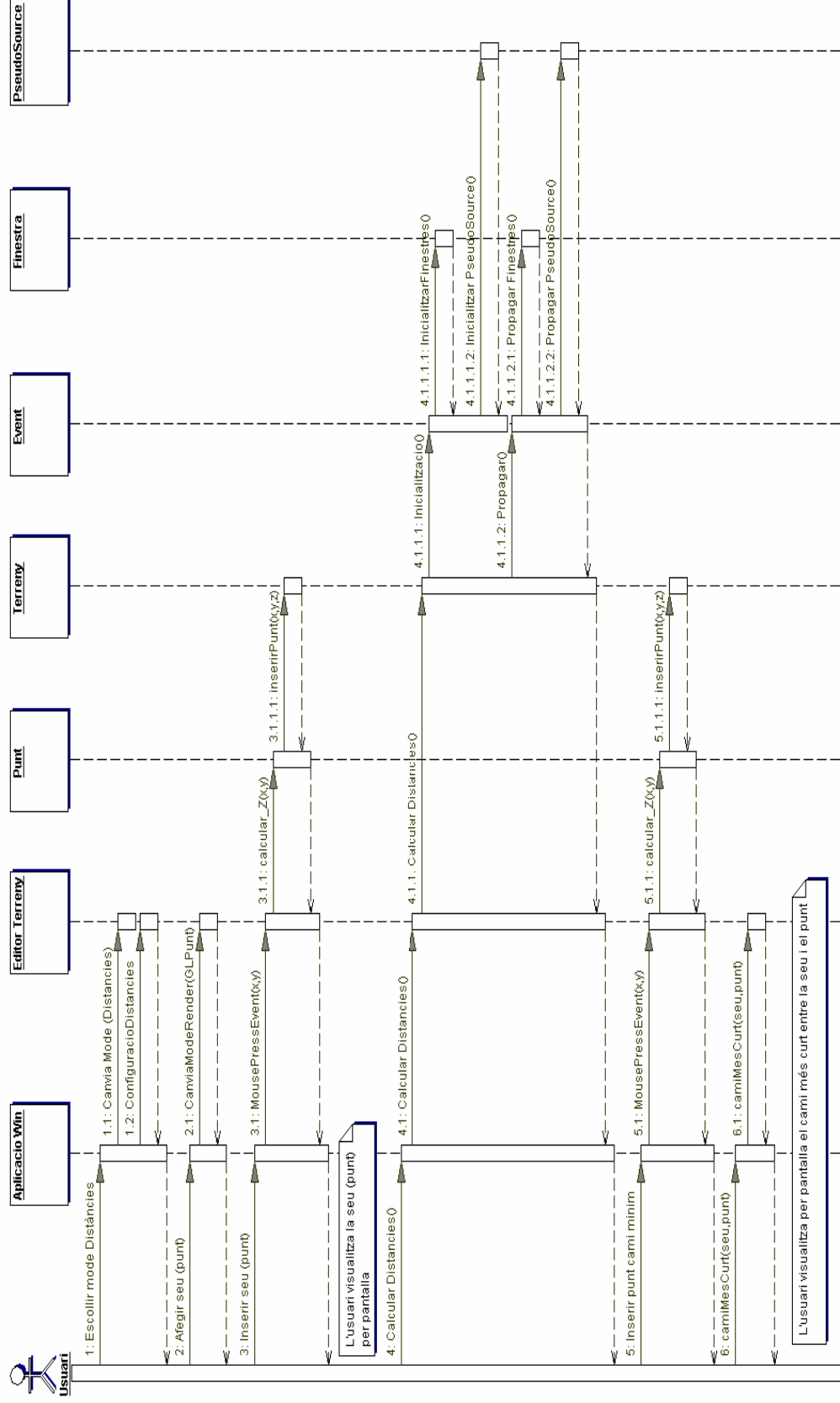


Figura 4.3 : Diagrama de seqüència Camí òptim entre seu i punt

4.2.3 Diagrama de seqüència Diagrama de Voronoi sobre Terreny

Una altra de les operacions més complexes i importants que ha de fer la nostra aplicació és el càlcul i la visualització del *Diagrama de Voronoi* sobre un terreny que disposa d'un conjunt de seus.

Tal i com es pot observar en la *Figura 4.4*, l'usuari interactua amb el sistema a través de la classe *Aplicacio Win*. Aquest envia les peticions a la classe de control *Editor Terreny* encarregada de distribuir cadascuna de les operacions a cadascuna de les entitats amb la capacitat de resoldre-les.

Suposant que ja estem treballant amb un terreny que hem generat anteriorment, els primers passos que caldrà seguir són els mateixos que hem fet per inserir una seu (diagrama de seqüència de la *Figura 4.2*), tot i que en aquest cas anirem inserint tantes seus com considerem oportú en el nostre terreny. (Nombre seus ≥ 2)

Un cop ja tinguem la seu inserida en el terreny, premerem una opció que ens cridarà un mètode de la classe *Editor Terreny* que ens realitzarà, a grans trets, la inicialització de finestres i pseudosources, la posterior propagació de finestres i pseudosources (tenint en compte també si les finestres es solapen),... És a dir, ens anirà posant finestres al llarg de tot el terreny

Finalment cridarem al mètode *veureDiagramaVoronoi()* que serà el que ens permetrà visualitzar el resultat final per la pantalla.

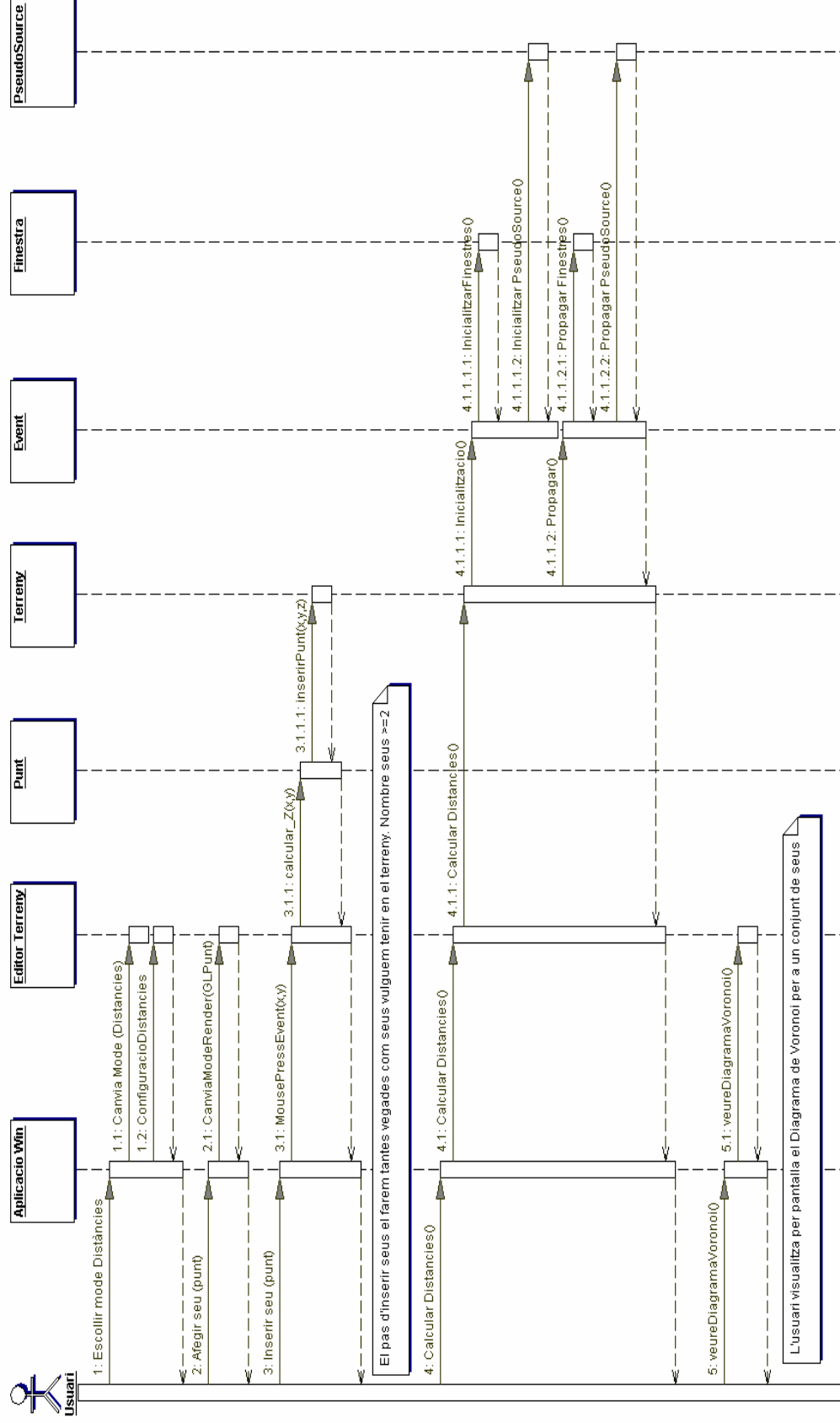


Figura 4.4 : Diagrama de seqüència Diagrama de Voronoi sobre terreny

Capítol 5

Disseny del Sistema

Un cop realitzades les fases *d'anàlisi de requeriments* i *d'anàlisi del sistema*, en aquest capítol veurem principalment quin ha estat el disseny de l'aplicació. A partir dels resultats de l'anàlisi, l'etapa de **disseny del sistema** s'ocupa de crear una solució conceptual que permeti implementar els requeriments del sistema a construir. És l'etapa anterior a la implementació del sistema i, per aquest motiu, cal definir els objectes de disseny que podran ésser directament implementables. Així doncs, en aquest capítol veurem principalment quin ha estat el disseny de l'aplicació, el disseny de la interfície gràfica i quins canvis s'hi han efectuat per tal de poder resoldre els nostres objectius del projecte.

5.1 Disseny de la interfície

En aquest apartat veurem quina ha estat la tecnologia utilitzada per tal d'implementar la **interfície**. Aquesta informació serà bastant importat a l'hora d'implementar la nostra aplicació. Tal i com ja havíem comentat anteriorment, per a la realització d'aquest projecte s'ha partit de la interfície gràfica realitzada en el projecte ***“Disseny d'una eina de modelat de terrenys i càlculs de corbes de nivell”*** i en el projecte ***“Algorismes de generació i modificació de terrenys”***. Aquesta interfície va ésser creada a partir de les eines que ofereixen les llibreries *Qt* i que ja hem comentat el seu funcionament en els fonaments pràctics del *Capítol 2* de la memòria. Aquestes llibreries, disposen d'una àmplia col·lecció de components gràfics i visuals que faciliten la tasca a l'hora de crear interfícies.

Podem dividir bàsicament la interfície en dues parts, l'**Editor 2D** i el **Visor 3D**. L'*Editor 2D* serà qui ens permetrà realitzar totes aquelles operacions pròpies de l'edició de terrenys, càlcul de distàncies, càlcul de camins òptims, càlcul dels Diagrames de Voronoi, ... de forma interactiva. Ens mostrarà la triangulació del terreny a tractar i ens permetrà modificar-lo (inserir seus, inserir punts de camí mínim, eliminar seus i punts de camí mínim, calcular el que hem comentat anteriorment, ...). D'altra banda, el *Visor 3D* serà qui ens permetrà representar el terreny triangulat en 2.5D amb el color i/o textures escollides. Aquesta finestra només serà de visualització i en tot moment reflectirà instantàniament tots els canvis que efectuem en l'*Editor 2D*. Així doncs, podem dir que a través del *Visor 3D* no podran realitzar-se modificacions sobre el terreny. Com que volem que tots els canvis que efectuem en l'*Editor 2D* es vegin reflectits també en el *Visor 3D*, requerirem que tots dos vegin el mateix objecte, cosa que ha estat possible gràcies a la utilització del patró **Singleton**.

En la realització d'aquest projecte, la interfície ha estat adaptada tot eliminant funcionalitats pròpies dels altres projectes, afegint-hi funcionalitats noves pròpies del nostre projecte o bé canviant el comportament de funcionalitats ja existents adequant-les a la nostra problemàtica. Gairebé tots els canvis que hem introduït s'han realitzat en la finestra de l'*Editor 2D*, que tal i com ja hem comentat, és la finestra amb la qual més interactuarà l'usuari de l'aplicació i allà on es realitzen tots els procediments i totes les opcions principals del nostre projecte. La distribució dels nous controls afegits dins de la finestra de l'aplicació s'han dut a terme seguint una certa lògica de processos. S'ha tingut també en compte la comprensió del funcionament de cadascuna de les operacions perquè qualsevol tipus d'usuari sigui capaç d'utilitzar l'aplicació. Del *Visor 3D* hem aprofitat bàsicament la carcassa i hi hem introduït els canvis pertinents per tal d'adequar-lo també a la nostra problemàtica.

La tecnologia utilitzada per tal de dissenyar la interfície han estat les llibreries **Qt**, utilitzant l'eina **QtDesigner**. Les llibreries que *Qt* aporta permeten al programador construir de manera molt senzilla una sèrie de controls, a través dels anomenats *widgets*, per tal de poder crear una interfície gràfica d'usuari agradable i fàcil d'utilitzar per a qualsevol tipus d'usuari. A més a més, la utilització d'aquest tipus d'interfície ens permetrà treballar a través de la programació orientada a events, de manera que segons les accions que realitzi l'usuari es realitzarà una o altra tasca. També ens permet detectar events relacionats amb perifèrics d'entrada i sortida, com poden ésser el teclat, el ratolí o el monitor. També ens permet treballar amb *signals* i *slots*. Una funció *signal* s'executa quan es compleix una determinada condició. Cada *signal* pot éstar connectat a una funció *slot* a través de la funció *connect*, de manera que si es genera un determinat *signal* provocarà que l'*slot* connectat a aquest s'executi.

A continuació anem a observar quina forma té la interfície final utilitzada tant de l'*Editor 2D* com del *Visor 3D*, comentant cada funcionalitat dels diferents menús i/o botons i explicant els principals *widgets* nous que s'han introduït per tal de proporcionar i facilitar la interacció entre l'usuari i el sistema.

5.1.1 Editor 2D

Tal i com ja hem comentat anteriorment, *l'Editor 2D* és la finestra principal de l'aplicació a través de la qual l'usuari realitzarà la interacció amb l'aplicació. La majoria de les funcionalitats permeses en el nostre projecte seran accessibles des de la finestra de *l'Editor 2D*. Així doncs, ens permetrà editar un terreny, inserir seus, realitzar el càlcul de camins òptims, visualitzar els Diagrames de Voronoi, ...

En la *Figura 5.1* podem observar quina és la forma que té la interfície final del programa pel que fa a *l'Editor 2D*.

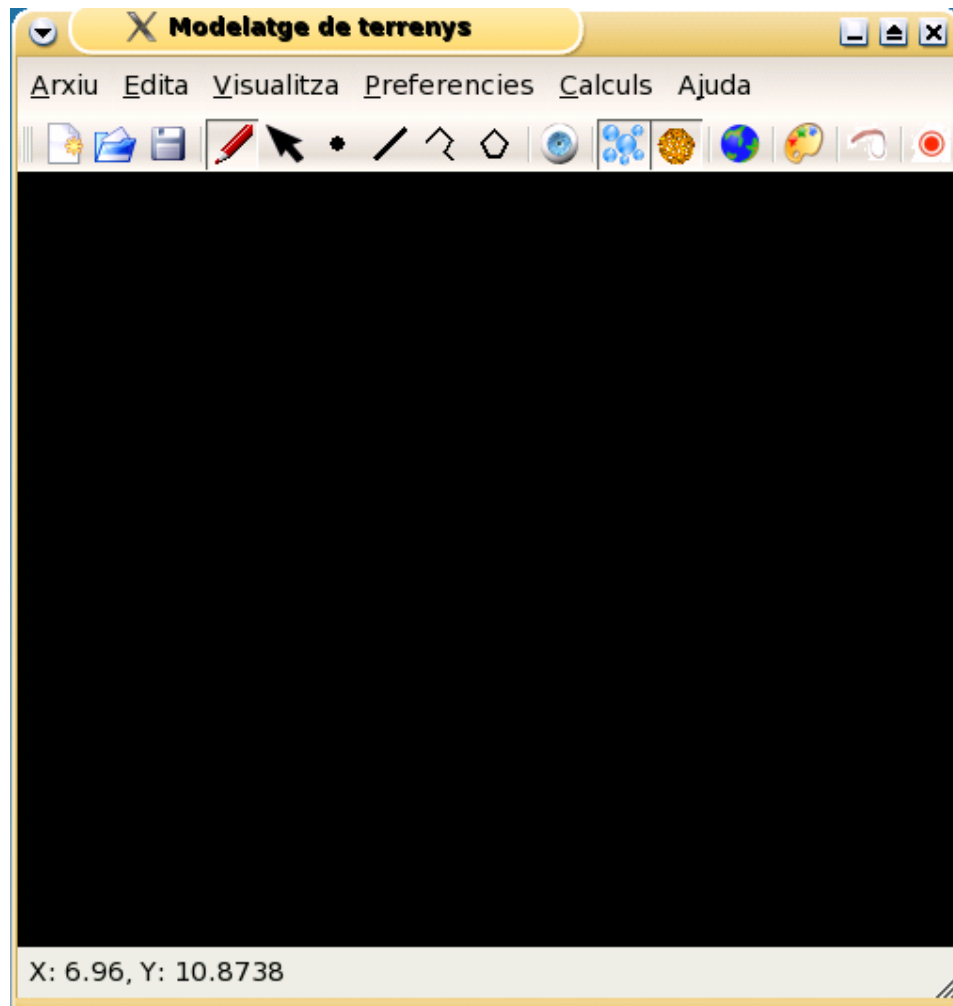




Figura 5.1 : Interfície final de l'Editor 2D

Tal i com es pot observar en la Figura anterior, la interfície de l'Editor 2D disposa d'un gran nombre de botons i de menús, que permetran a l'usuari en tot moment realitzar totes les operacions que desitgi en l'Editor 2D. A continuació anem a descriure de quins menús es disposa, quines opcions o submenús hi ha dins de cada menú i quins botons ens permeten realitzar les mateixes opcions que ens ofereixen els menús o altres opcions a les quals no s'hi pot accedir via menú.

Nota: Només comentarem les opcions que s'utilitzen per tal de resoldre la problemàtica que afecta al nostre projecte. Les opcions que ja tenia la interfície de l'Editor 2D en la seva versió inicial i que no intervenen per res en el nostre projecte, no es tindran en compte.



→ Menú Arxiu

- *Nou*
 - *Nou Terreny* 
 - *Nou Terreny Aleatori*
- *Carrega terreny* 
- *Desa terreny* 
- *Exporta a imatge*
- *Surt*

→ Menú Edita

- *Mode Edició* 
- *Mode Selecció* 
- *Inserir Punt (seu)* 
- *Inserir Segment* 
- *Inserir Poligonal* 
- *Inserir Polígon* 
- *Inserir Punt Terreny* 


→ Menú Visualitza

- *Mostra Terreny*
- *Elements visibles* 
- *Diagrames de Voronoi* 

→ Menú Preferències

- *Configura ...*

→ Menú Càlculs

- *Visibilitat*
- *Distàncies*
- *Trobar camí més curt*
- *Propagar Terreny* 

→ Menú Ajuda

- *Quant a*
- *Quant a QT*
- *Què és això*

5.1.2 Visor 3D

Tal i com ja havíem comentat anteriorment, el *Visor 3D* és la finestra que fa la funció de visualitzador del model del terreny en l'espai 3D. Podrà ésser o no visualitzada segons es seleccioni la opció corresponent de la anterior finestra principal, tal i com veurem més endavant. Des d'aquesta finestra, l'usuari serà capaç de veure el terreny des de diferents perspectives.

En la *Figura 5.2* de la pàgina següent podem observar quina és la forma que té la interfície final del programa pel que fa a la finestra del *Visor 3D*.

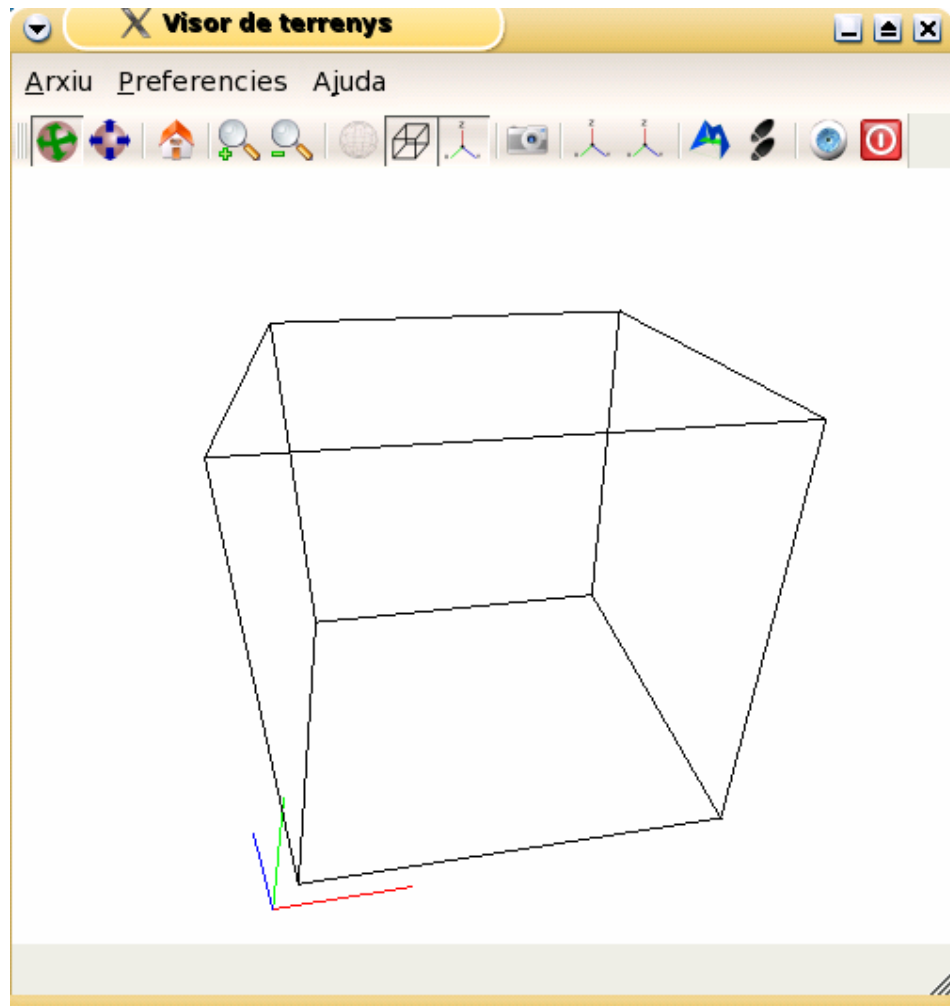



Figura 5.2 : Interfície final del Visor 3D

Tal i com es pot observar en la *Figura 5.2*, la interfície del *Visor 3D* disposa d'una sèrie de botons i de menús, que permetran a l'usuari en tot moment realitzar totes les operacions que desitgi en el *Visor 3D*. A continuació anem a descriure de quins menús es disposa, quines opcions o submenús hi ha dins de cada menú i quins botons ens permeten realitzar les mateixes opcions que ens ofereixen els menús o altres opcions a les quals no s'hi pot accedir via menú.

Nota: *Només comentarem les opcions que s'utilitzen per tal de resoldre la problemàtica que afecta al nostre projecte. Les opcions que ja tenia la interfície del Visor 3D en la seva versió inicial i que no intervenen per res en el nostre projecte, no es tindran en compte.*

→ Menú Arxiu

- *Captura Imatge* 




→ Menú Preferències

- *Configura Visor ...*

→ Menú Ajuda

- *Quant a*
- *Quant a QT*
- *Què és això*

→ Funcionalitats de botons (que no es troben a cap menú)

- *Rotar càmera* 
- *Desplaçar càmera* 
- *Iniciar vista (Home)* 
- *Apropar càmera (+)* 
- *Allunyar càmera (-)* 
- *Veure límits de l'espai* 
- *Veure eixos de coordenades* 
- *Finestres* 
- *Elements Visibles* 
- *Surt del programa* 

5.2 Format dels fitxers del terreny

Una de les funcionalitats de l'aplicació, tal i com ja havíem comentat anteriorment, ha d'ésser la funcionalitat de poder carregar terrenys generats prèviament (i que els haguem emmagatzemat) i el fet de poder-los emmagatzemar en un format que pugui reconèixer l'aplicació. Dit d'una altra manera, hem de poder gestionar la persistència del nostre sistema. També havíem de poder capturar imatges tant en 2D com en 3D del terreny, tal i com hem especificat en alguns dels casos d'ús descrits en el *Capítol 3: "Requeriments del Sistema"*.

Així doncs, l'aplicació ha de tenir la capacitat de guardar i recuperar la informació referent a la triangulació del terreny. La nostra aplicació suportarà 3 formats diferents :

- **Format Geonview .off**

El format **.off** (*Object File Format*) és propi de **Geonview**, que és un programa interactiu de visió 3D per a Unix, el qual permet veure i manipular objectes tridimensionals. També pot ésser utilitzat com un visualitzador independent o com a motor gràfic per a altres aplicacions que es basin en la realització de models geomètrics, ja que permet treballar amb objectes tridimensionals guardats en una gran varietat de formats.

- **Format DTM**

El format **DTM** és un format estàndard a partir del qual podem carregar terrenys reals a la nostra aplicació. Els models de terrenys digitals són la representació dels punts d'elevació de la superfície terrestre. Aquests són utilitzats per generar models de superfície i contorns, o per el procés d'ortorectificació de fotografia aèria.

- **Format propi .tps**

El format **.tps** és un format propi de l'aplicació, que ens permet guardar tota la informació relacionada amb el sistema. D'una banda podem obtenir tota la informació referent a la triangulació del terreny (*DCEL*) i de l'altra tota la informació referent als elements restringits (*PSLG*) que puguem inserir a la nostra triangulació.

5.3 Format dels fitxers d'imatge

En el nostre programa serà important que l'usuari pugui guardar imatges del terreny per tal de poder-les visualitzar sempre que vulgui. Així doncs, l'usuari haurà de poder desar els resultats obtinguts tant des del model 2D com des del model 3D en un format d'imatge. Per fer-ho, es disposa d'una opció que en qualsevol moment ens permet realitzar una captura de la imatge (que sempre tindrà una qualitat del 100%) com si es tractés d'una fotografia, la qual posteriorment podrà ésser editada amb qualsevol visor d'imatges extern.

L'usuari podrà escollir entre diferents formats (amb diferent extensió) sempre i quant el sistema tingui suport per a aquests. Els tipus de **formats dels fitxers d'imatge** disponibles en el nostre sistema són els següents:

Format fitxers d'imatge							
.bmp	.jpg	.pbm	.png	.xpm	.xbm	.ppm	.pgm

Capítol 6

Implementació

En aquest capítol explicarem com hem realitzat la fase d'implementació en aquest projecte. Després d'haver realitzat l'anàlisi de requeriments, l'anàlisi del sistema i el disseny del sistema, passarem a realitzar la implementació. Aquest procés serà iteratiu tal i com indica la metodologia orientada a objectes que s'ha utilitzat.

La implementació d'aquesta aplicació s'ha realitzat a través del llenguatge de programació C++, el qual ens ha aportat diferents conceptes útils per tal de millorar la distribució i eficiència del programa com poden ésser la utilització de classes, objectes, herència, ... També ens ha permès unificar les rutines de les llibreries *OpenGL* i *Qt* que havíem utilitzat.

A continuació doncs veurem quins són els algorismes més importants que hem implementat, classificant-los segons les seves funcionalitats i segons quins problemes de proximitat en terrenys presenten (determinar camins òptims, calcular la distància geodèsica, calcular diagrames de Voronoi, ...). A més a més, per a cada algorisme implementat veurem alguns exemples visuals d'imatges generades amb l'algorisme corresponent.

6.1 Càlcul camins òptims

El que volem realitzar, és poder obtenir la reconstrucció del camí mínim que hi ha entre un punt del terreny i una seu. Per a realitzar-ho, hem decidit implementar l'algorisme de trobar camins mínims en un terreny basant-nos en les directrius que s'expliquen en la referència [SSKGH05] sobre l'algorisme d'ordre $O(n^2 \log n)$ proposat per Mitchell [MMP87].

Per a poder realitzar el càlcul d'aquest camí mínim i poder-lo reconstruir posteriorment, el primer que ens caldrà és disposar d'un terreny amb una seu inserida, tal i com es pot observar a la *Figura 6.1*

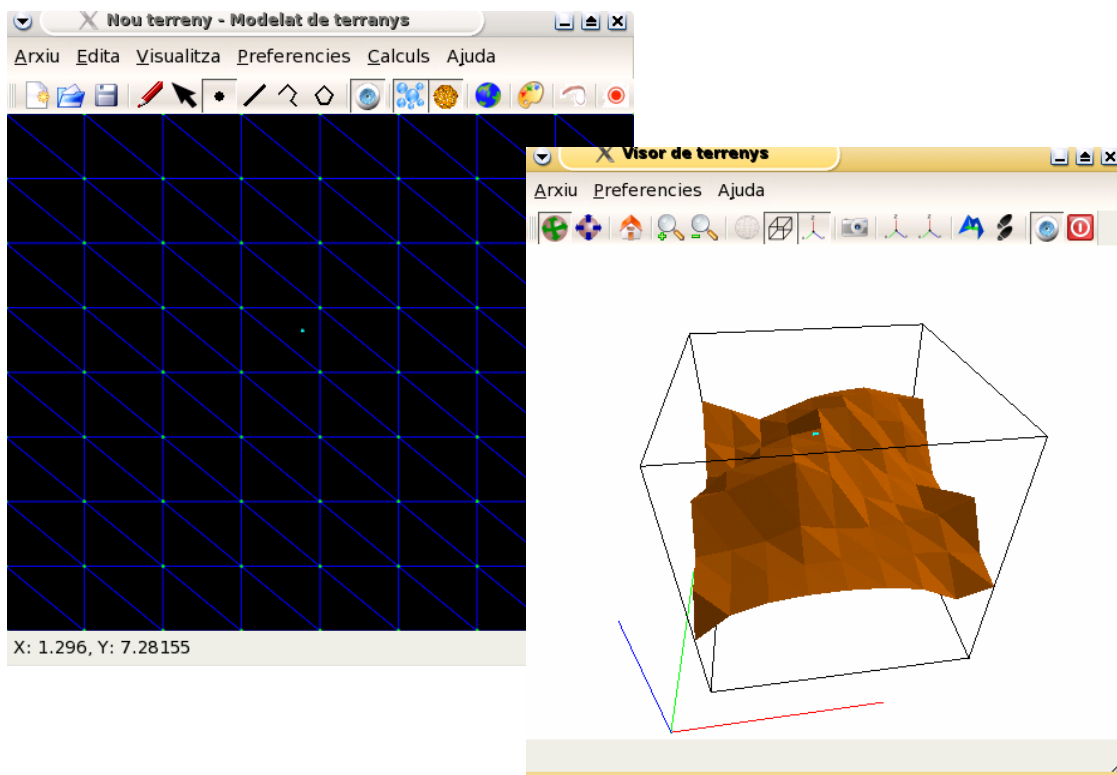


Figura 6.1 : Terreny amb una seu inserida

Els camins mínims són visualitzats com a rajos que surten d'un vèrtex seu v_s en totes les direccions tangents de la superfície polièdrica P al vèrtex v_s . Es pot realitzar la caracterització dels camins mínims en superfícies triangulars no convexes amb tres propietats:

- 1 → Dins d'una cara d'un triangle el camí més curt és una línia recta
- 2 → Quant travessa un costat, el camí més curt correspon a una línia recta quan dues cares adjacents es fan coplanars
- 3 → Els camins mínims només poden passar per un vèrtex quan aquest és un vèrtex hiperbòlic o de sella ($>2PI$)

La funció de distància del camí més curt definida per un punt seu s , és una funció D_s tal que per a cada punt q de la superfície polièdrica P , $D_s(q)$ és la llargada del camí més curt de q a la seu s . Així doncs, el camí més curt de la seu s a qualsevol altra punt destí q tindrà la caracterització que acabem de descriure.

La idea bàsica, consisteix en agrupar els camins mínims d'una manera que puguin ésser parametritzats de forma automàtica. Amb aquest objectiu, els costats es parteixen en una sèrie d'interval·ls anomenats **Finestres**. Una altra estructura que haurem de emmagatzemar, són els **PseudoSources** o seus puntuals, que seran els que es formaran quan realitzem la propagació de finestres, i que sempre seran vèrtexs de la triangulació del terreny. Tractarem un nou pseudosource quan ens trobem un vèrtex del terreny que sigui hiperbòlic. Per tal d'emmagatzemar aquestes dues estructures de dades, s'ha creat una classe general abstracta anomenada **Event**, on hi haurà definits tots els mètodes utilitzats a les classes **Finestra** i **PseudoSource**. D'aquesta manera, tindrem un atribut a la classe *Event* anomenat *tipus_event* que ens indicarà si aquell *event* és de tipus *Finestra* o és de tipus *PseudoSource*. Així doncs les classes *Finestra* i *PseudoSource* tindran la següent caracterització:

Classe Finestra	Classe PseudoSource
double b0	double sigma
double b1	pVertex vertex
double d0	pCara cara
	double d1
	double sigma
	pCara cara
	int j

De la **Finestra** ens guardarem els paràmetres (bo , $b1$, $d0$, $d1$, $sigma$, $cara$, j), i del **PseudoSource** ens guardarem els paràmetres ($sigma$, $vertex$, $cara$). Per a més informació sobre aquests paràmetres veure els fonaments teòrics del *Capítol 2*.

Una altra cosa que necessitarem és guardar a cada vèrtex del terreny la distància a la seu. Per realitzar això, en el fitxer de definicions *geometriabasica.h* definirem la següent estructura:

```
struct InfoVertex
{
    double distancia;
};
```

Primerament hem inicialitzat tots els vèrtexs del terreny a distància infinita, amb el mètode *inicialitzaVertexs()*. Aquesta distància l'anirem actualitzant sempre que trobem un camí que ens porti a aquell vèrtex amb una distància més petita que la distància actual. Després de inicialitzar els vèrtexs, ja estarem en condicions de realitzar el pas d'inicialització, amb el mètode

inicialitzacio(), que ens crearà (inserirà) finestres als costats del triangle en què es troba la seu, i ja ens les anirà inserint a una cua de prioritats ordenada per pes (de menys a més), agafant $pes = \min(d0, d1) + \sigma$. També, si ens trobem amb algun vèrtex hiperbòlic (hem realitzat una funció a la classe *Terreny* anomenada *vertexHiperbolic(cara, vertex)* que ens ho mira), inserirem el nou *PseudoSource* a la cua de prioritats amb el seu corresponent pes, que en el cas del *PseudoSource* serà: $pes = \sigma$. En la *Figura 6.2* podem observar com en el pas d'inicialització ja s'han col·locat les primeres finestres als costats del triangle on es troba la seu.

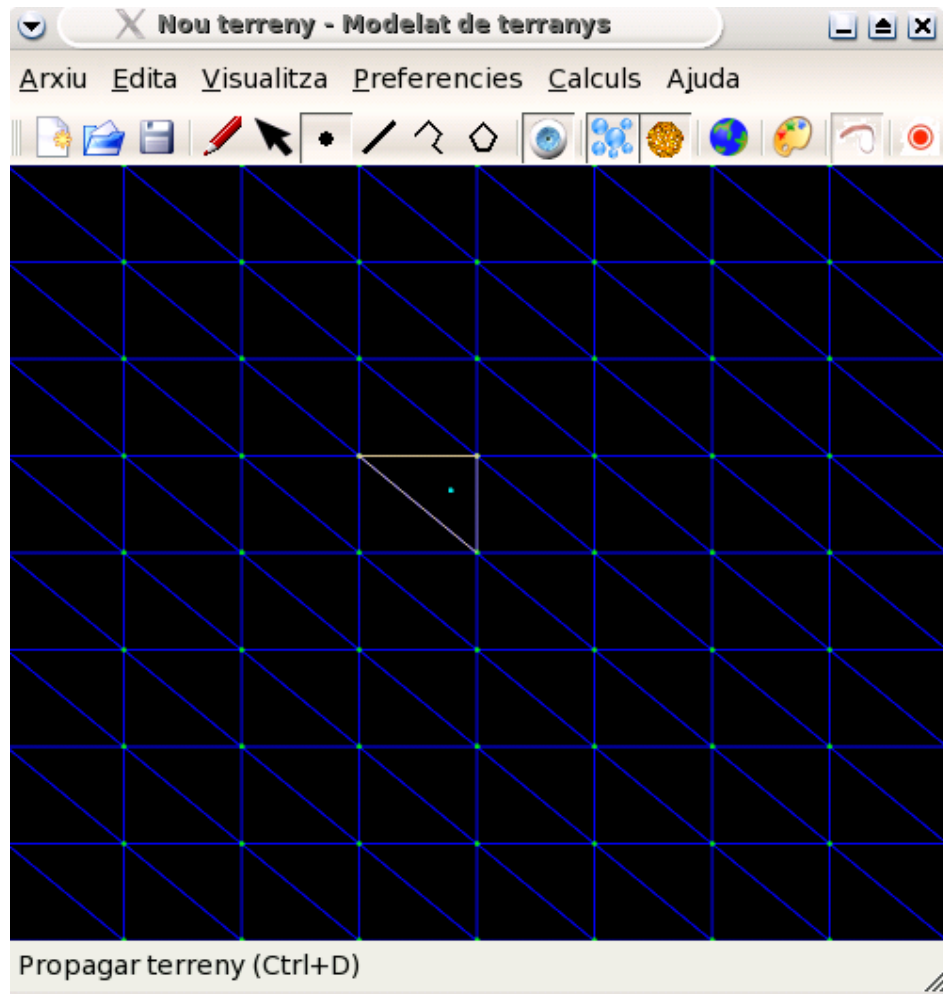


Figura 6.2 : Finestres creades a partir del pas d'inicialització

Un cop hem realitzat el mètode de **Inicialització**, ja disposarem **d'events** (*Finestres* i/o *PseudoSources*) a la cua de prioritats ordenats per pes. El que ens caldrà fer posteriorment és, mentre la cua de prioritats no sigui buida, anar propagant cadascun d'aquests *events*, mitjançant el mètode *Propagació(cua_de_prioritat)*. Cada **event** que anem propagant ena anirà generant noves *Finestres* i nous *PseudoSources*. Els *PseudoSources* ja els inserirem directament a la cua de prioritats a la posició que els hi correspongui segons el seu respectiu pes (mètode *propagarPseudoSource()*), mentre que les *Finestres* (mètode *propagarFinestra()*) no les inserirem directament a la

cua, sinó que abans hauran de passar per el mètode *deteccióEventsReals(Llista_de_finestres_possibles)*, que farà que no inserim finestres repetides a la cua de prioritat i ens realitzarà tota la problemàtica referent al fet de si ens trobàvem finestres en un mateix costat (tant si miràvem del costat de la cara com del costat de la cara_veïna) que es solapaven o intersecaven, que s'havien de retallar i tornar a recalcular els extrems ($b0, b1$) i les distàncies ($d0, d1$) tal i com hem explicat en l'apartat de fonaments teòrics del capítol 2. A la *Figura 6.3* podem observar un exemple de com es van propagant les *Finestres* i els *PseudoSources* al llarg del terreny.

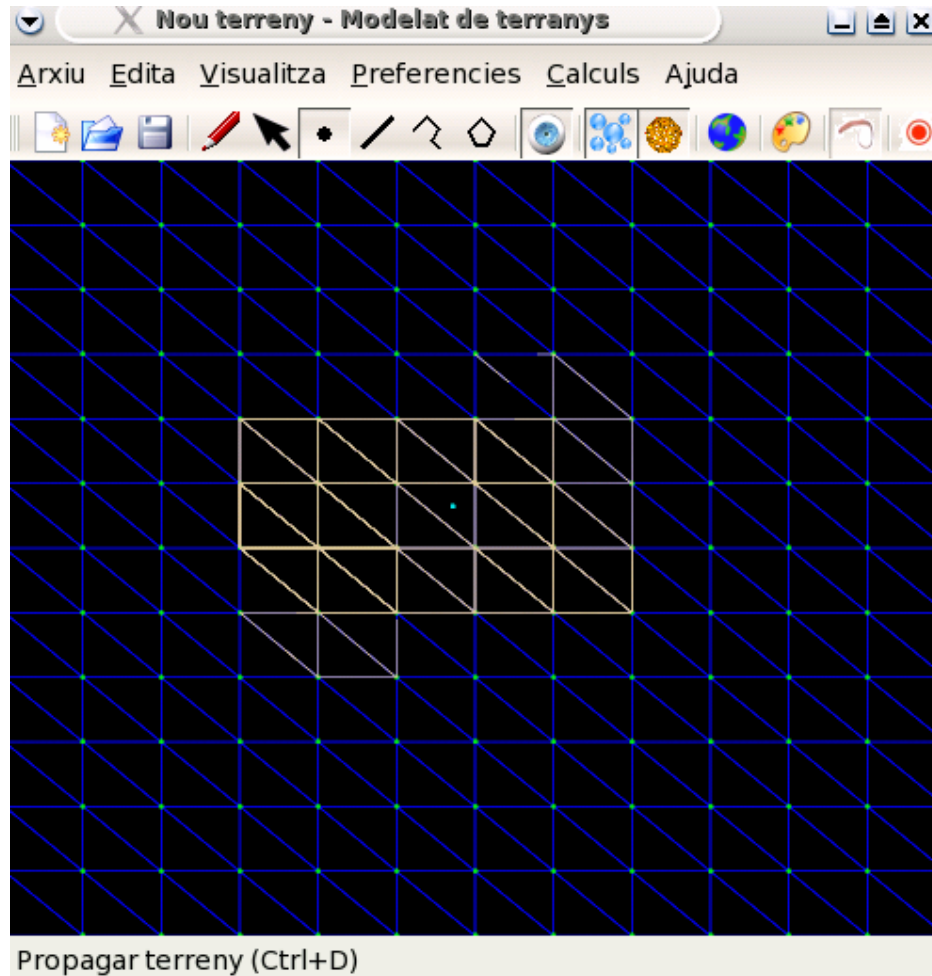


Figura 6.3 : Propagació de finestres i pseudosources al llarg del terreny

Tal i com es pot observar en la *Figura 6.3*, podem observar amb color groc les finestres que es van creant i propagant als costats de la triangulació, a partir de la seu inserida, al llarg de tot el terreny. Arribats a aquest punt, ja hauré propagat totes les finestres al llarg del terreny, és a dir, tindrem tots els costats dels triangles del terreny coberts de finestres. Llavors, serà el moment d'inserir un punt qualsevol al terreny (que no actuï com a seu), que serà el punt del qual voldrem saber la distància respecte a la seu. La inserció d'aquest punt, que es veurà d'un color diferent (groc) a la seu, es pot observar en la *Figura 6.4*.

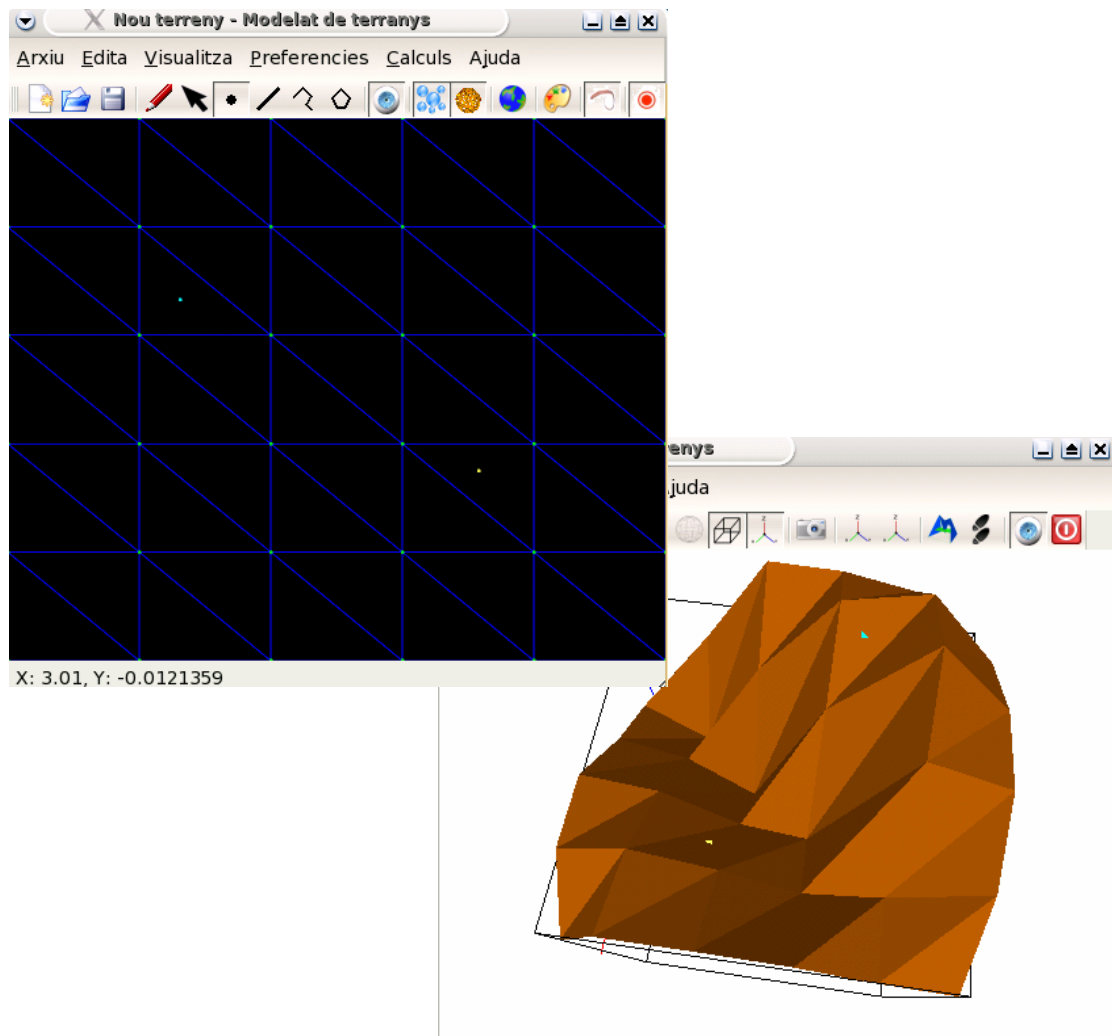


Figura 6.4 : Punt de camí mínim (color groc) inserit en el terreny

Arribats a aquest punt, un cop ja disposem de la seu i del punt del qual volem trobar el camí mínim, realitzarem una tècnica de **Backtracing** per tal de reconstruir el camí mínim que hi ha entre la seu i el punt que acabem d'inserir. Per més detalls sobre com funciona aquesta tècnica, mirar el que hem explicat en els fonaments teòrics del *Capítol 2*. En les següents *Figures 6.5, 6.6 i 6.7* podem observar exemples visuals de la reconstrucció del camí mínim entre una seu i un punt. En la finestra de *l'Editor 2D* podem observar de color vermell el camí mínim que ens ha trobat l'algorisme entre la seu (de color verd) i el punt de camí mínim (de color groc). En la finestra del *Visor 3D* podem observar aquest mateix camí d'un color groguenc, però visualitzat en 3 dimensions,

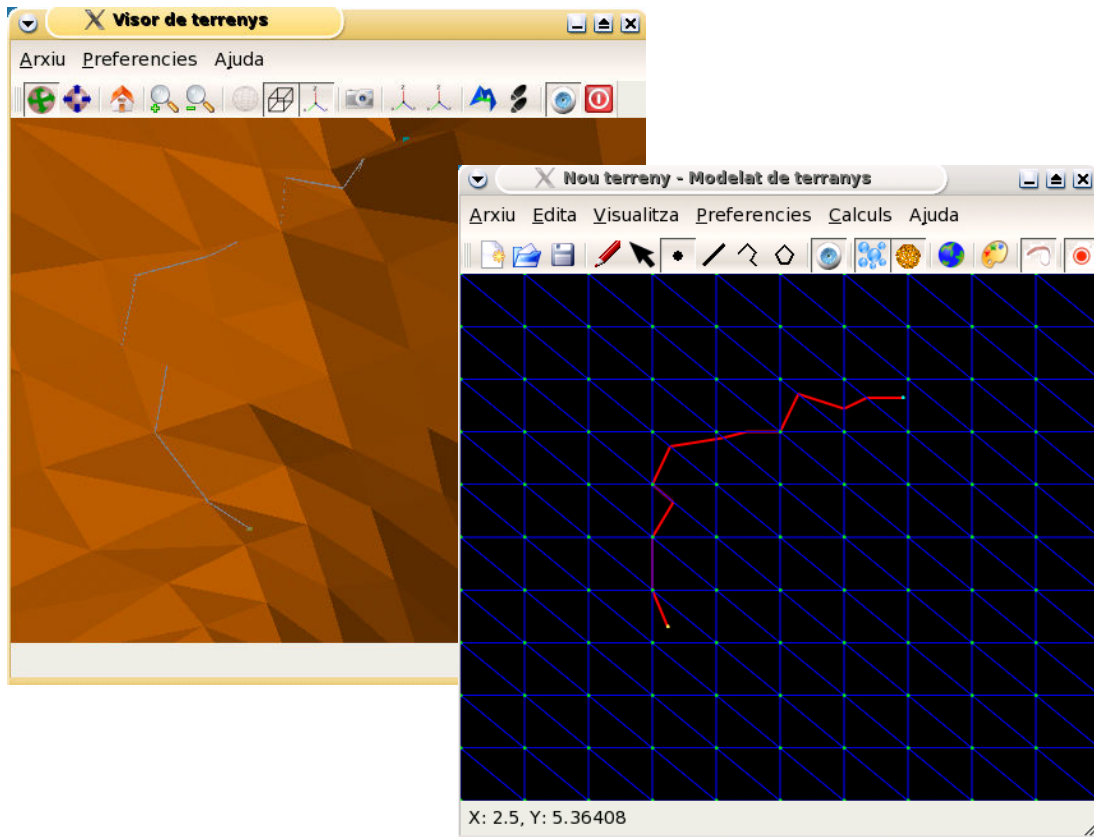


Figura 6.5 : Exemple camí més curt entre seu i punt

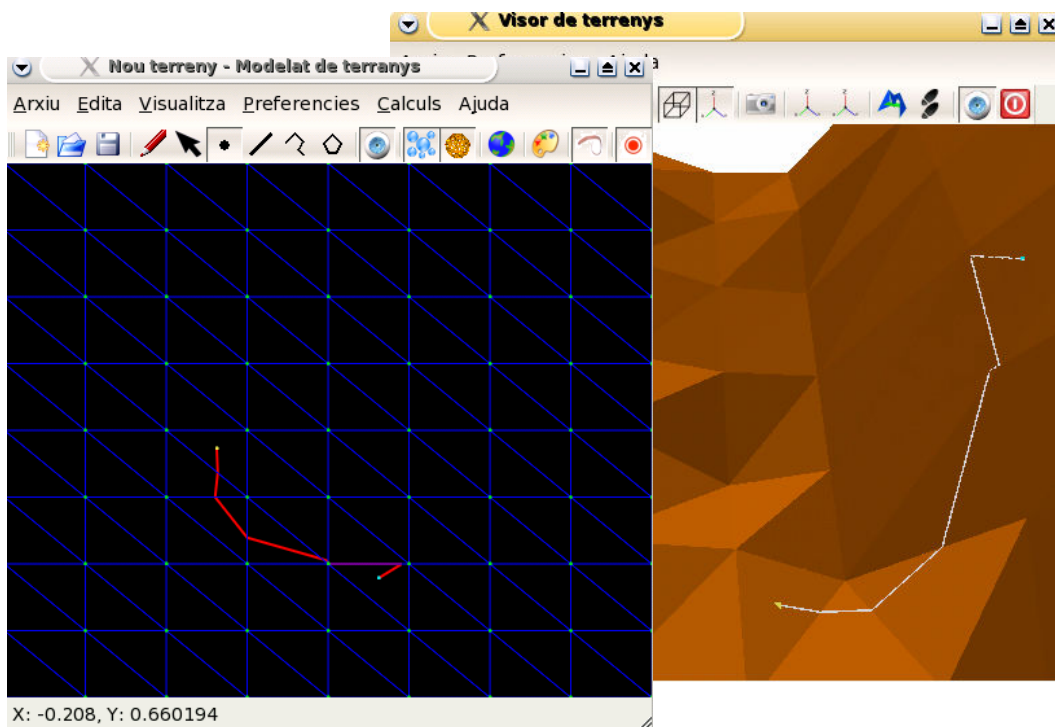


Figura 6.6 : Exemple camí més curt entre seu i punt

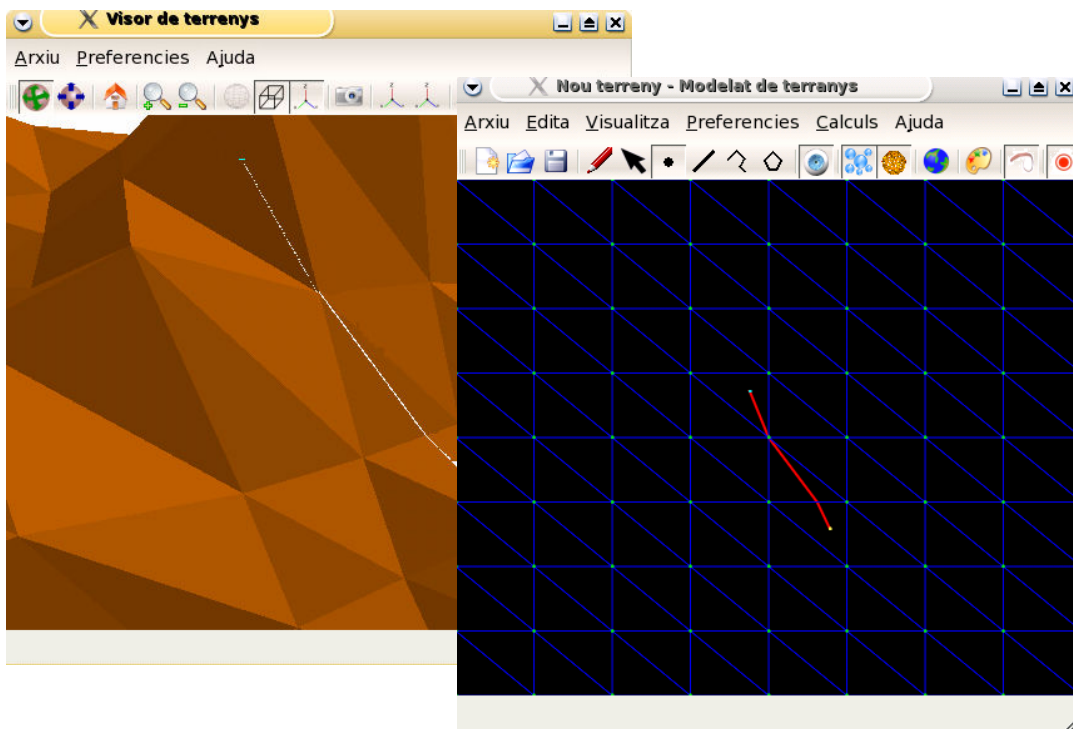


Figura 6.7 : Exemple camí més curt entre seu i punt

6.2 Diagrames de Voronoi

Tal i com s'ha pogut observar en els exemples visuals anteriors, tots els exemples que hem realitzat disposaven només d'una única seu. És a dir, que l'algorisme que hem fet servir en l'apartat anterior per tal d'obtenir el camí mínim entre una seu i un punt s'haurà de fer extensible al cas d'un terreny que disposi d'un conjunt de seus (veure *Figura 6.8*). Serà aquí quant entraran en joc els **Diagrames de Voronoi**, que recordem que eren la estructura més típica per tal de representar termes de proximitat relacionats amb una sèrie de punts o seus en la Geometria Computacional. L'objectiu serà que per a cada punt ens doni la distància mínima a la seu més propera.

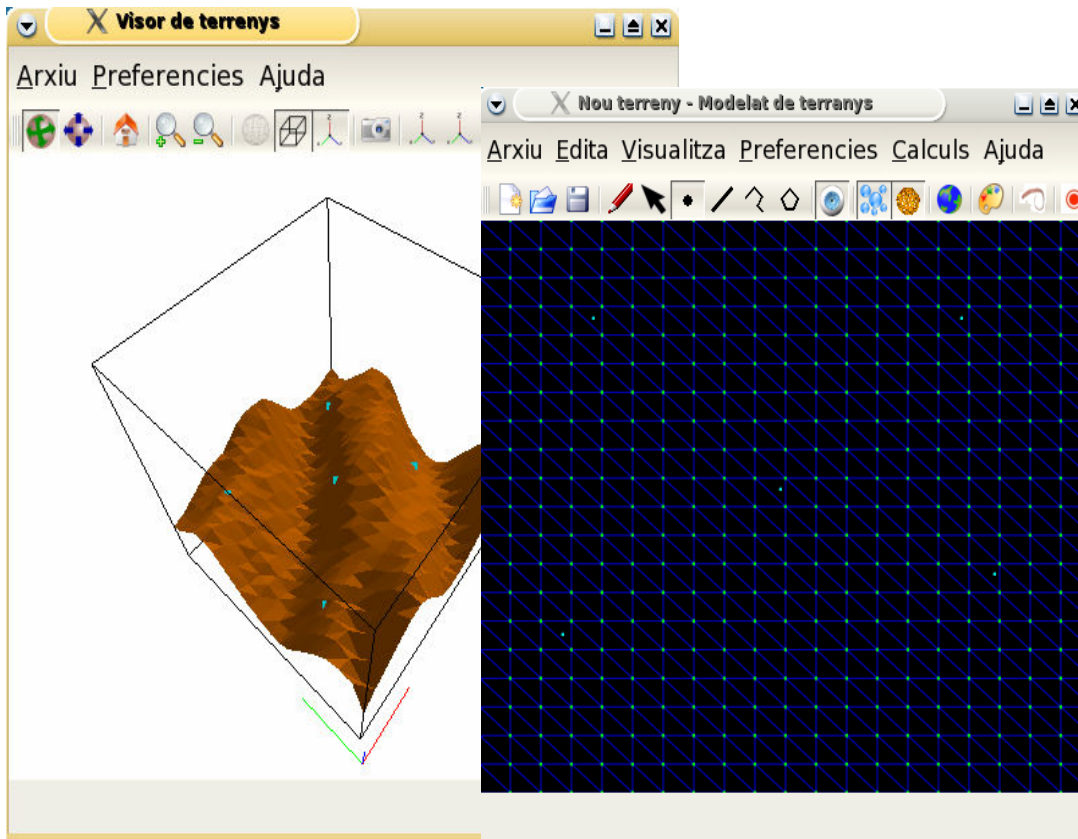


Figura 6.8 : Terreny amb més d'una seu

Així doncs, per tal d'obtenir el *Diagrama de Voronoi* d'un conjunt de seus $O=\{s_1, \dots, s_k\}$, utilitzarem l'algorisme que hem fet per tal de calcular la funció de distància a la seu més propera. Això es durà a terme considerant totes les seus en el pas d'inicialització i inicialitzant la llista (cua de prioritat) amb totes les finestres. Per això, serà interessant conèixer de quina seu prové cada finestra o cada pseudosource nou que anem creant i propagant. Amb aquesta finalitat, afegirem un nou atribut a l'estructura de la classe **Finestra** i a la de la classe **PseudoSource**, anomenat **índex**, que anirà de 1 fins a k (essent k el nombre de seus), que serà l'índex de la seu de la qual la finestra prové.

Així doncs les classes *Finestra* i *PseudoSource* tindran ara la següent caracterització:

Classe Finestra	Classe PseudoSource
double b0	double sigma
double b1	int index
double d0	pVertex vertex
double d1	pCara cara
	double sigma
	pCara cara
	int j
	int index

D'aquesta manera, propaguem simultàniament el camp de distàncies definit per cada una de les diferents seus. Automàticament podrem obtenir una codificació de la funció de distàncies generalitzada que ens produeix una representació implícita del diagrama de Voronoi del conjunt de seus.

Per tal de visualitzar una aproximació del Diagrama de Voronoi, calcularem una aproximació del camp de distàncies utilitzant només la distància als vèrtexs del terreny i hardware gràfic. Durant aquest procés, necessitarem guardar a cada vèrtex del terreny la distància a la seu més propera i l'índex d'aquesta seu. Per realitzar això, en el fitxer de definicions *geometriabasica.h* ampliarem la següent estructura:

```
struct InfoVertex
{
    double distancia;
    int index;
};
```

Un cop aconseguim aquesta informació, una aproximació discreta del **Diagrama de Voronoi** es pot obtenir utilitzant hardware gràfic, fent ús de la inherent interpolació bilineal durant el procés de rasterització. Per tal d'obtenir-lo, utilitzarem els següents mètodes de la classe *EditorTerreny*:

```
void EditorTerreny::dibuixarDiagramesVoronoi()
void EditorTerreny::mostrarDiagramaVoronoi()
void EditorTerreny::propagacioCaraAproximada(pCara cara)
void EditorTerreny::activarFragment()
void EditorTerreny::assignarColorDV(int j)
```

El Diagrama de Voronoi s'obté utilitzant *OpenGL* i agafant els avantatges que ens produeix el Test de Profunditat (*Depth Test*). Per això en el mètode *mostrarDiagramaVoronoi()* direm que volem treballar amb el Test de Profunditat.

```
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_GEQUAL);
glColor4f(0,0,0,1);
glBegin(GL_QUADS);
    glVertex3d(info_mon.xmin,info_mon.ymin,-10);
    glVertex3d(info_mon.xmin,info_mon.ymax,-10);
    glVertex3d(info_mon.xmax,info_mon.ymax,-10);
    glVertex3d(info_mon.xmax,info_mon.ymin,-10);
glEnd();
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LEQUAL);
```

Anirem pintant un darrera l'altre els camps de distàncies de cada seu mitjançant colors diferents. Aquests colors s'assignaran de forma aleatòria en el mètode *assignarColorDV(int j)* on li passarem el paràmetre enter *j*, que serà l'índex de la seu. Si pintem el mínim valor de la coordenada Z, obtindrem la mínima distància de cada punt respecte al conjunt de seus. Quant totes les seus hagin estat tractades, la imatge obtinguda en el “*frame buffer*” (*activarFragment()*) és el Diagrama de Voronoi, i els valors guardats en el “*buffer*” de profunditat (*z-buffer*) és el camp de distància associat al conjunt de seus.

```
activarFragment();
cgGLEnableProfile(info_vis.gh.fragmentProfile);

glBegin( GL_TRIANGLES );

vector<pCara> v_cares=mesh->getCares();
for(int j=0; j<v_cares.size(); j++)
    propagacioCaraAproximada(v_cares[j]);

glEnd();
```

Per tal de representar la funció de distància i obtenir el Diagrama de Voronoi, pintarem les cares del terreny amb una funció aproximada de distància amb el color associat a la seu que defineix la funció de distància. Això ho realitzarem en el mètode *propagacioCaraAproximada(pCara cara)*, on en els fonaments teòrics del *Capítol 2* ja hem comentat que podíem diferenciar-hi dos casos diferents:

- Cares amb els 3 vèrtexs amb la mateixa seu propera
- Cares amb vèrtexs amb diferents seus properes

A continuació anem a veure exemples visuals de l'aplicació utilitzant un conjunt de seus i obtenint els corresponents **Diagrames de Voronoi**.

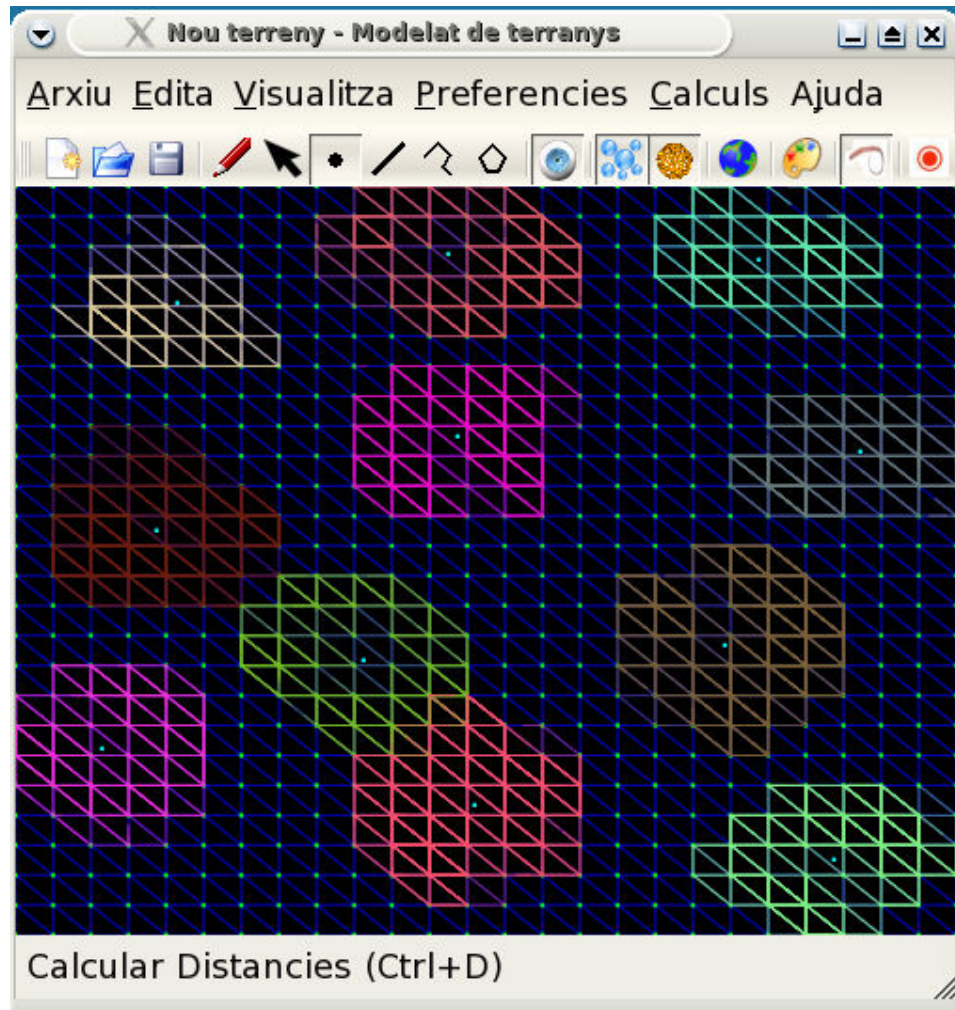


Figura 6.9 : Exemple propagació de finestres per a un conjunt de seus

En la *Figura 6.9* podem observar com hem inserit un conjunt de seus (punts) en el terreny , i al escollir la opció *Propagar Terreny* del menú *Calculs* hem anat creant i propagant les finestres d'aquest conjunt de seus al llarg del terreny. Podem observar com cada seu crea finestres d'un color diferent, que seran els mateixos colors que posteriorment es visualitzaran en el *Diagrama de Voronoi* i que ens permetran saber per cada punt qualsevol del terreny quina és la seu que està més a prop seu.

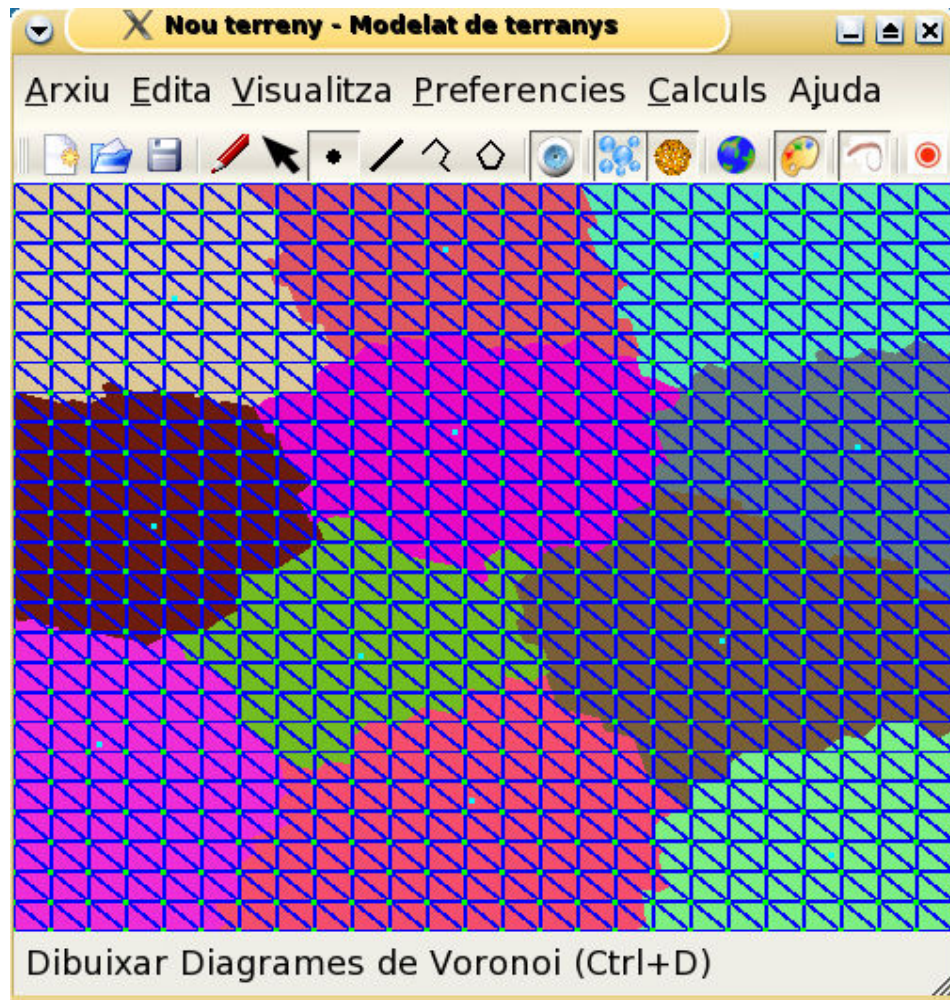


Figura 6.10 : Diagrama de Voronoi per al conjunt de seus de l'exemple anterior

En la *Figura 6.10* podem observar després d'escollir la opció *Diagrames de Voronoi* del menú *Visualitza* quin és el Diagrama de Voronoi per al conjunt de seus que hem vist com es propagaven en la *Figura 6.9* anterior. D'aquesta manera, tots els punts del terreny que estan sota la superfície d'un mateix color, voldrà dir que la seu que tenen més a prop és la seu que està continguda també dins la superfície amb aquell color (totes les seus són de color verd en el dibuix). Així doncs, donat un punt qualsevol del terreny sabrem dins quina regió del **Diagrama de Voronoi** el podem englobar i quina és la seu que té més propera.

En les *Figures 6.11* i *6.12* podem observar dos nous exemples de **Diagrames de Voronoi** per a un conjunt de 5 i 3 seus respectivament.

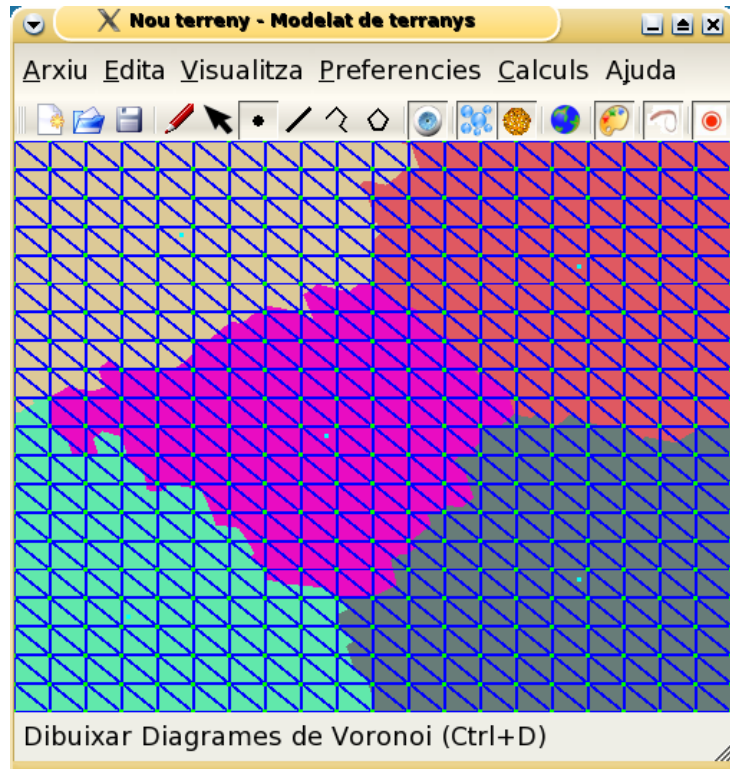


Figura 6.11 : Diagrama de Voronoi per a un conjunt de 5 seus

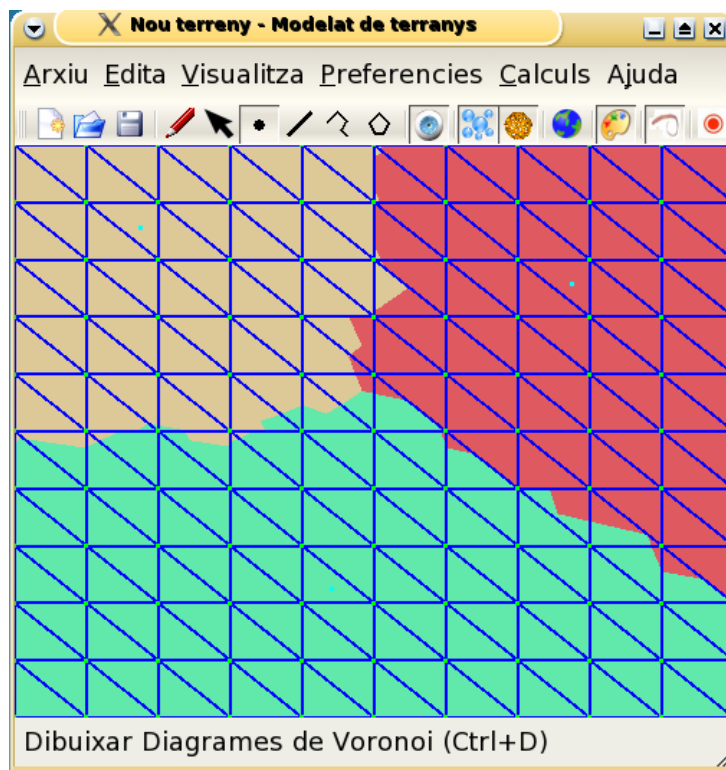


Figura 6.12 : Diagrama de Voronoi per a un conjunt de 3 seus

6.3 Jocs de proves

Un fet important que no podem deixar de banda a l'hora de parlar de la implementació de la nostra aplicació, és el fet que per tal de poder realitzar amb èxit tots els algorismes referents al camí òptim entre una seu i un punt i a la visualització de Diagrames de Voronoi per a un conjunt de seus, ha calgut realitzar nombrosos jocs de proves per tal de testejar cadascuna de les funcions (mètodes) més importants del programa, per tal de poder garantir de forma correcta el funcionament de la nostra aplicació. Tots aquests jocs de proves (que en la seva majoria s'han realitzat des de la classe *EditorTerreny*) ens han portat un bon temps de preparació, però han estat imprescindibles per tal de poder corregir algun error i per tal d'obtenir finalment el correcte funcionament de l'aplicació.

A continuació anem a descriure quines són les principals funcions (mètodes) de les quals s'han realitzat jocs de proves i, si és el cas, il·lustrarem amb imatges visuals o amb resultats numèrics el resultat obtingut mitjançant tots aquests jocs de proves:

- **Trobar Source**
- **Pas d'inicialització**
- **Propagar PseudoSource**
- **Propagar Finestra**
- **Interseca**
- **No_Elimina**
- **Actualitzar Llista Finestres**

Trobar Source

Donats per teclat els paràmetres d'una finestra $(b0, b1, d0, d1)$, afegint també el fet de si es tractava d'una finestra d'una cara veïna o d'una cara no_veïna, i si era veïna indicant també la mida e del costat, la funció de testeig *Trobar Source* ens retornava les coordenades de la seu s , així com la distància del punt $(b0, 0)$ a la seu i la distància del punt $(b1, 0)$ a la seu.

```

***** TESTEIG DE FUNCIONS *****
1 --> Trobar Source
2 --> Inicialització
3 --> Propagar PseudoSource
4 --> Propagar Finestra
5 --> Interseca
6 --> No_Elimina
7 --> Actualitzar
*****

Escriu una opció : 1

Parametres finestra *****
b0 = 0.6
, b1 = 0.9
, d0 = 0.2
, d1 = 0.5
, veins = (1=true,0=false) 0
, e = 1

Source s = ( 0.4 , 6.08124e-09 )
Distancia del punt (b0,0) al source = 0.2
Distancia del punt (b1,0) al source = 0.5
Fi testeig del Trobar Source

```

Figura 6.13 : Joc de proves de la funció Trobar Source

Pas d'inicialització

Donat un terreny generat de forma aleatòria o bé creat per l'usuari, amb una o diverses seus inserides en el terreny, la funció de testeig *Pas d'Inicialització* ens realitzava el pas d'inicialització, és a dir, ens posava finestres a cadascun dels costats del triangle que contenia la seu, per d'aquesta manera, podem començar a realitzar la posterior propagació de finestres.

```

***** TESTEIG DE FUNCIONS *****
1 --> Trobar Source
2 --> Inicialització
3 --> Propagar PseudoSource
4 --> Propagar Finestra
5 --> Interseca
6 --> No_Elimina
7 --> Actualitzar
*****

Escriu una opció : 2
Hem inicialitzat les distàncies dels vertexs a -1

Punt = 3.824 , 4.58252 , 12.0341
Triangle que conté el source = 4 5 13.131 , 3 5 13.2013 , 4 4 10.4738
Creem la finestra amb parametres d0 = 1.48849 d1 = 1.67472 b0 = 0 b1 = 3.07229 sigma = 0 index = 1 j = 0
Puntvertex 1 = 3 , 5 , 13.2013
Puntvertex 2 = 4 , 4 , 10.4738
Inserim a la llista d'Events la finestra amb j = 0 i amb pes = 1.48849
Creem la finestra amb parametres d0 = 1.67472 d1 = 1.18684 b0 = 0 b1 = 2.83915 sigma = 0 index = 1 j = 1
Puntvertex 1 = 4 , 4 , 10.4738
Puntvertex 2 = 4 , 5 , 13.131
Inserim a la llista d'Events la finestra amb j = 1 i amb pes = 1.18684
Creem la finestra amb parametres d0 = 1.18684 d1 = 1.48849 b0 = 0 b1 = 1.00246 sigma = 0 index = 1 j = 2
Puntvertex 1 = 4 , 5 , 13.131
Puntvertex 2 = 3 , 5 , 13.2013
Inserim a la llista d'Events la finestra amb j = 2 i amb pes = 1.18684
Pes Primer Element Llista = 1.18684 Pes Ultim Element Llista 1.48849
Pes Primer Element Llista = 1.18684 Pes Ultim Element Llista 1.48849
Hem realitzat el mètode de inicialització

```

Figura 6.14 : Joc de proves de la funció Inicialització

Propagar PseudoSource

Donat un terreny generat de forma aleatòria o bé creat per l'usuari, amb una o diverses seus inserides en el terreny, ens realitzava el pas d'inicialització (funcionalitat que acabem de comentar en la funció de testeig anterior) i posteriorment ens propagava només els *PseudoSources* que hi havia en el terreny, és a dir, de la llista *d'events*, només ens tenia en compte els que eren de tipus *PseudoSource*.

```

EVENT: PS 0.453521 (sigma, index, pv) = (0.453521, 1, (4, 5, 12.5376) )
Troblem un PseudoSource amb pes 0.453521
F 0.453521 (0, 1.0465, 0, 1.0465, 0.453521, 0, 1)
F 1.50002 (0, 1.2925, 1.0465, 1.50349, 0.453521, 1, 1)
F 0.453521 (0, 1.50349, 0, 1.50349, 0.453521, 0, 1)
F 1.45692 (0, 1.08768, 1.50349, 1.0034, 0.453521, 1, 1)
F 0.453521 (0, 1.0034, 0, 1.0034, 0.453521, 0, 1)
F 1.45692 (0, 1.46907, 1.0034, 1.04849, 0.453521, 1, 1)
F 0.453521 (0, 1.04849, 0, 1.04849, 0.453521, 0, 1)
F 1.50201 (0, 1.24841, 1.04849, 1.47877, 0.453521, 1, 1)
F 0.453521 (0, 1.17418, 1.17418, 0, 0.453521, 0, 1)
F 0.453521 (0, 1.47877, 0, 1.47877, 0.453521, 1, 1)

```

Figura 6.15 : Joc de proves de la funció Propagar PseudoSource

Propagar Finestra

Donat un terreny amb una seu inserida i una finestra, amb els seus corresponents paràmetres, el mètode ens realitza la propagació d'aquesta finestra, dibuixant-nos la seu original, la seu puntual originada per aquesta finestra, la finestra original i les noves finestres que es formen en el/s costat/s de la cara veïna una vegada hem propagat la finestra original.

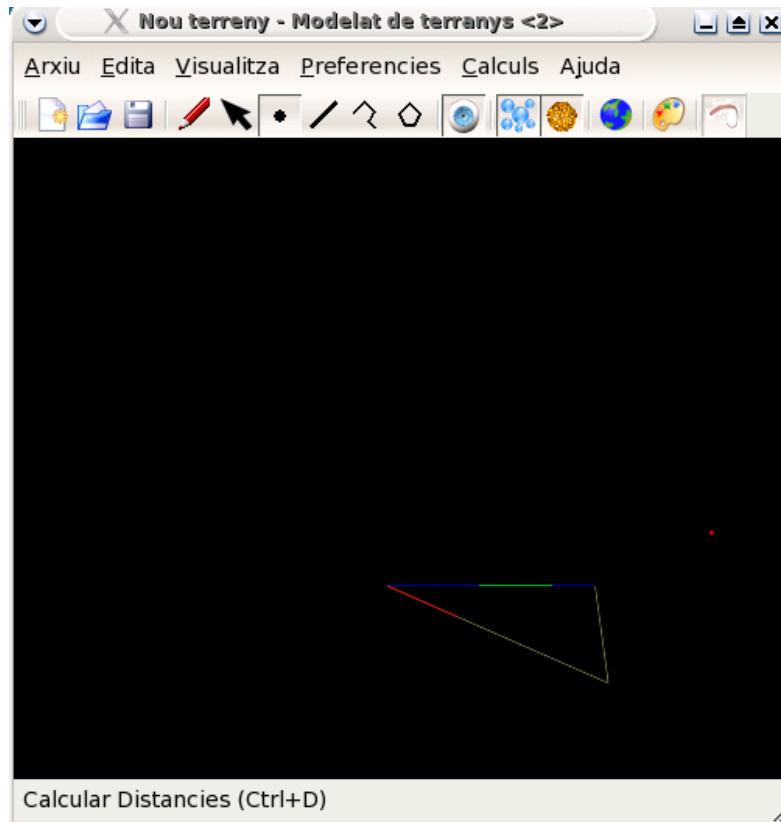


Figura 6.16 : Joc de proves de la funció Propagar Finestra

Interseca

Donades per teclat dues finestres, amb els seus corresponents paràmetres ($b_0, b_1, d_0, d_1, \sigma$) per cadascuna d'elles i els paràmetres *veïna* i *e*, si era el cas, ens retornava si les dues finestres intersecaven o no, i en cas que interseguessin, ens retornava el punt on intersecaven i la distància del punt equidistant a la seu.

```

Parametres finestra 1 *****
b0 = 0.3
, b1 = 0.9
, d0 = 1.2
, d1 = 1.5
, sigma = 0

Finestra 1 creada!!

Parametres finestra 2 *****
b0 = 0.2
, b1 = 0.7
, d0 = 0
, d1 = 0.5
, sigma = 1

Finestra 2 creada!!
, veins = (1=true,0=false) 0
, e = 1
Source Finestra f1 = -0.075 , 1.1399
Source Finestra f2 = 0.2 , 5.55112e-17
----- Agafo x=-B-sqrt(B^2-4AC))/(2A) (A,B,C) = ( -0.924375 , 0.472875 , -0.0224438 )
VALORS QUE RETORNA : x = 0.458621 , d = 1.25862
METODE DISTANCIA SOURCE TROBAT = -0.075 , 1.1399
METODE DISTANCIA SOURCE TROBAT = 0.2 , 5.55112e-17
Distancia de la finestra 1 a x = 1.25862
Distancia de la finestra 2 a x = 1.25862
LES FINESTRES INTERSEQUEN!!

```

Figura 6.17 : Joc de proves de la funció Interseca

No Elimina

Donades per teclat dues finestres, amb els seus corresponents paràmetres ($b_0, b_1, d_0, d_1, \sigma$) per cadascuna d'elles i els paràmetres *veïna* i *e*, si era el cas, ens retornava si les dues finestres intersecaven o no, i en cas que intersequessin, ens retornava el punt on intersecaven i la distància del punt equidistant a la seu. Posteriorment, si intersecaven, el que es realitzava en la funció de testeig *No_Elimina* era la actualització d'aquestes finestres d'acord amb la seva intersecció i el seu punt d'intersecció, retornant-nos els nous paràmetres que tindrien, tant dels seus extrems (els nous b_0' i b_1' per cadascuna d'elles) com de les distàncies (els nous d_0' i d_1' per cadascuna d'elles)

```

Finestra f1 abans actualitzar : ( 0.3 , 0.9 , 1.2 , 1.5 , 0 )
Finestra f2 abans actualitzar : ( 0.2 , 0.7 , 0 , 0.5 , 1 )
METODE DISTANCIA SOURCE TROBAT = -0.075 , 1.1399
METODE DISTANCIA SOURCE TROBAT = 0.2 , 5.55112e-17
Distancia de f1 a x = 1.25862
Distancia de f2 a x = 1.25862
NO VEINS: IF
METODE DISTANCIA SOURCE TROBAT = 0.2 , 5.55112e-17
Finestra f1 despres actualitzar : ( 0.458621 , 0.9 , 1.25862 , 1.5 , 0 )
Finestra f2 despres actualitzar : ( 0.2 , 0.458621 , 0 , 0.258621 , 1 )
Valor no_elimina = (1-true,0-false) 1

```

Figura 6.18 : Joc de proves de la funció No_Elimina

Actualitzar Llista Finestres

Donades per teclat diverses finestres, amb els seus corresponents paràmetres ($b_0, b_1, d_0, d_1, \sigma, j, index, cara$) per cadascuna d'elles, cadascuna de les quals s'inseria en una llista d'*events*, el mètode de testeig *Actualitzar Llista Finestres* ens mirava si alguna d'aquestes finestres era igual a alguna ja existent en la llista d'*events*. En cas afirmatiu, ens esborrava una de les dues finestres que eren iguals de la llista d'*events*.

```

Mida llista = 2
Event ev = F 0.5 (0.3, 0.9, 0.5, 0.6, 0, 1, 1)
eliminem 1 event
Mida llista = 1

```

Figura 6.19 : Joc de proves de la funció Actualitzar Llista Finestres

Capítol 7

Millores i Treball futur

Recordem el fet que tots aquests algorismes i tècniques de resolució de problemes estan integrats en una interfície que també disposa d'altres funcionalitats apart de les que hem aportat en aquest projecte. Hem intentat dissenyar una aplicació que fos extensible per tal de poder afegir-hi noves funcionalitats en un futur. Som conscients que a partir d'ara caldrà continuar fent un bon treball per tal de millorar la interfície i fer que aquesta pugui oferir més prestacions als usuaris.

A continuació explicaré les que crec que són les millores que estaria bé que es poguessin realitzar en la part referent al càlcul de distàncies, càlcul/reconstrucció i visualització de camins òptims i càlcul i visualització de Diagrames de Voronoi per a un conjunt de seus, amb les quals es continuarà treballant en un futur proper per millorar les prestacions que ofereix tota aquesta part referent al càlcul de distàncies de la interfície.

- **Visualitzar Diagrames de Voronoi en 3D**

Un aspecte que seria interessant que ens oferís la interfície, seria la possibilitat de poder visualitzar també els Diagrames de Voronoi per a un conjunt de seus en el Visor 3D, ja que actualment només es visualitzen en l'editor 2D.

- **Utilitzar segments, polígons i poligonals com a seus**

Un altre aspecte, que ja hem remarcat diverses vegades durant la realització de la memòria d'aquest projecte, és fer operativa la part en què puguin actuar com a seus altres figures geomètriques que no siguin punts. Així doncs, hauríem de fer operatiu que es poguessin inserir segments, polígons i poligonals que actuessin com a seus i es pogués realitzar el mateix tipus d'operacions que amb les seus que són punts.

- **Poder calcular la distància mínima entre diferents tipus de seus i diferents tipus d'elements (punts, segments, polígons, poligonals,...)**

En aquest projecte sempre hem calculat el camí òptim considerant com a seus punts, i com a punts del terreny també punts. Estaria bé, que a part de poder també inserir com a seus segments, polígons i poligonals, tal i com acabem de comentar en la millora anterior, es pogués calcular la distància mínima (el camí òptim) entre diferents tipus de seus (punts, segments, polígons i poligonals) i diferents tipus d'elements (punts, segments, polígons i poligonals). És a dir, que tant les seus com els "punts" del terreny poguessin ésser qualsevol d'aquestes figures geomètriques.

Aquestes han estat, a grans trets, les millores que a dia d'avui crec que serien interessant per a la part referent al càlcul de camins òptims i visualització de Diagrames de Voronoi per a la interfície. Algunes d'aquestes millores ja s'intentarà començar-les a integrar a la interfície el més aviat possible, per tal de poder millorar les prestacions que aquesta ens ofereix. Altres millores que es podrien incorporar al programari, però que no fan referència exclusivament a la problemàtica de distàncies en terrenys podrien ésser:

- Afegir nous tipus d'elements de restricció, com poden ésser corbes o forats.
- Donar suport per a altres formats, simplement afegint un nou conversor entre aquest i la nostra aplicació
- Generar un terreny amb qualitat
- Aportar realisme al terreny (afegir vegetació (arbres, plantes, cultius, ...), elements artificials (edificis, ponts, ports, ...), ...)
- Disposar d'una opció per tal de desfer l'última acció realitzada en el terreny
- Suport per terrenys de mida considerable (>1.000.000 de punts) sense perdre rendiment ni velocitat

Capítol 8

Manual d'Usuari

En aquest capítol explicarem el manual d'usuari. A través d'aquest manual s'especifica el funcionament de cadascun dels elements de l'aplicació (tant menús com botons). D'aquesta manera, qualsevol usuari podrà utilitzar el programa simplement utilitzant aquest manual. A més a més, es troba acompanyat d'imatges que fan que tota l'explicació sigui molt més entenedora i pràctica.

Tal i com hem comentat, l'aplicació permetrà a l'usuari realitzar totes les funcionalitats que havíem descrit en l'etapa de requeriments del sistema de forma lògica i senzilla, a través d'una interfície amigable, intentant que l'usuari es senti còmode i segur a l'hora de treballar amb l'aplicació. Per tal d'aconseguir-ho, s'han utilitzat conceptes relacionats amb el disseny d'interfícies d'usuari.

Primer de tot explicarem els passos necessaris per tal de poder instal·lar el programari en qualsevol ordinador que compleixi els requeriments no funcionals descrits en el *Capítol 3*, i posteriorment explicarem, acompanyats d'imatges visuals, quines són les funcionalitats de les quals disposa el programa i quins passos cal fer per poder-les realitzar.

8.1 Instal·lació i execució del programari

Abans d'instal·lar el programari, haurem de disposar (i si no és el cas haurem d'instal·lar) aquelles llibreries que són necessàries per tal de poder compilar i executar el nostre programa de forma correcta. Així doncs, primer de tot hauríem de comprovar que tenim instal·lat el següent programari :

- Les biblioteques *Qt* versió 3.3.x i les eines de desenvolupament (*qmake, uic, moc, ...*)
- Les biblioteques de desenvolupament de l'entorn escollit, amb suport per *OpenGL* (llibreries Mesa) i acceleració 3D.
- Eines de desenvolupament (*g++, make, ...*)

Normalment la majoria de distribucions de la plataforma *Linux* ofereixen paquets precompilats de les biblioteques *Qt*, de les eines de desenvolupament (*g++, make, ...*) i les biblioteques de desenvolupament de l'entorn escollit, així com els drivers de les targetes gràfiques que permeten activar l'acceleració 3D d'aquestes, sempre i quan la targeta de l'ordinador ho suporti. En el cas que ens falti alguna d'aquestes coses, les distribucions disposen de programari que ens permet baixar-nos d'Internet aquests elements.

Un cop disposem de tots aquests elements que hem comentat, ja estarem en condicions de poder **compilar** i **executar** la nostra aplicació. Per tal de realitzar aquestes opcions, disposem de diverses opcions. Les més habituals seran:

- Si disposem del software de l'eina *KDevelop* (més informació sobre aquesta eina ens els fonaments pràctics del *Capítol 2*), ens serà molt fàcil realitzar la compilació i posterior execució de l'aplicació. El que haurem de fer és obrir el projecte amb aquest programa i triar la opció *Construir Proyecto* del menú *Construir*, la qual ens permetrà compilar tot el programa per tal que aquest pugui ésser executat (també disposem de la possibilitat de realitzar-ho amb el botó corresponent). A continuació, tenim la possibilitat de poder executar-lo des del mateix programa, a través de l'opció *Ejecutar programa principal* del menú *Construir* (també disposem de la possibilitat de realitzar-ho amb el botó corresponent)
- Una altra possibilitat consisteix en realitzar-ho a partir del fitxer projecte (fitxer amb extensió *.pro*) que es distribueix conjuntament amb el codi font. Aquest fitxer és generat a través de la comanda *qmake* amb la opció *-project*. Primerament ens haurem de situar al terminal i anar al directori de treball on tinguem el codi font i el fitxer projecte (*.pro*). Un cop arribats a aquest punt, ens caldrà seguir els següents passos:

- Executar la comanda *qmake*, la qual ens generarà un *makefile*.
- Executar la comanda *make*, la qual ens executarà el *makefile* i ens generarà l'executable en el directori *bin* de l'arbre de directoris del programari
- Introduir la comanda *./terrano* , la qual ens permetrà executar el programa. Una altra solució seria fer doble click sobre el fitxer executable. En cap dels 2 casos seran necessaris paràmetres addicionals per a la seva execució.

Arribats a aquest punt, ja podrem visualitzar la interfície de l'aplicació i ja estarem preparats per poder començar-hi a interactuar. En els següents apartats d'aquest capítol comentarem quina és la forma i els aspectes principals de la interfície i explicarem totes les funcionalitats que ens oferirà el programa a través dels menús i dels botons de què disposa.

8.2 Elements de la interfície

Per tal d'implementar la aplicació s'han tingut en compte els requeriments de programari especificats de bon principi en l'etapa d'anàlisi de requeriments. Per tal de poder aplicar els algorismes de càlcul de camins òptims i de visualització de Diagrames de Voronoi, ens cal una interfície senzilla d'entendre. La interfície ha estat pensada de tal manera que sigui amena i senzilla d'utilitzar per l'usuari (tenint en compte aspectes de l'assignatura *Interfícies d'Usuari*), de manera que la introducció de les dades i les modificacions i operacions en el terreny siguin senzilles i es puguin dur a terme sense gaire esforç per a la gran majoria d'usuaris. Per aquest motiu, la interfície s'ha dissenyat a través de dues finestres: l'**Editor 2D** i el **Visor 3D**

8.2.1 Editor 2D

És la finestra principal de l'aplicació a través de la qual l'usuari realitzarà la interacció amb l'aplicació. La majoria de les funcionalitats permeses en el nostre projecte seran accessibles des de la finestra de l'*Editor 2D*. Així doncs, ens permetrà editar un terreny, inserir seus, realitzar el càlcul de camins òptims, visualitzar els Diagrames de Voronoi, ...

En la *Figura 8.1* de la pàgina següent podem observar quina és la forma que té la interfície final del programa pel que fa a l'*Editor 2D* un cop executem el programa.

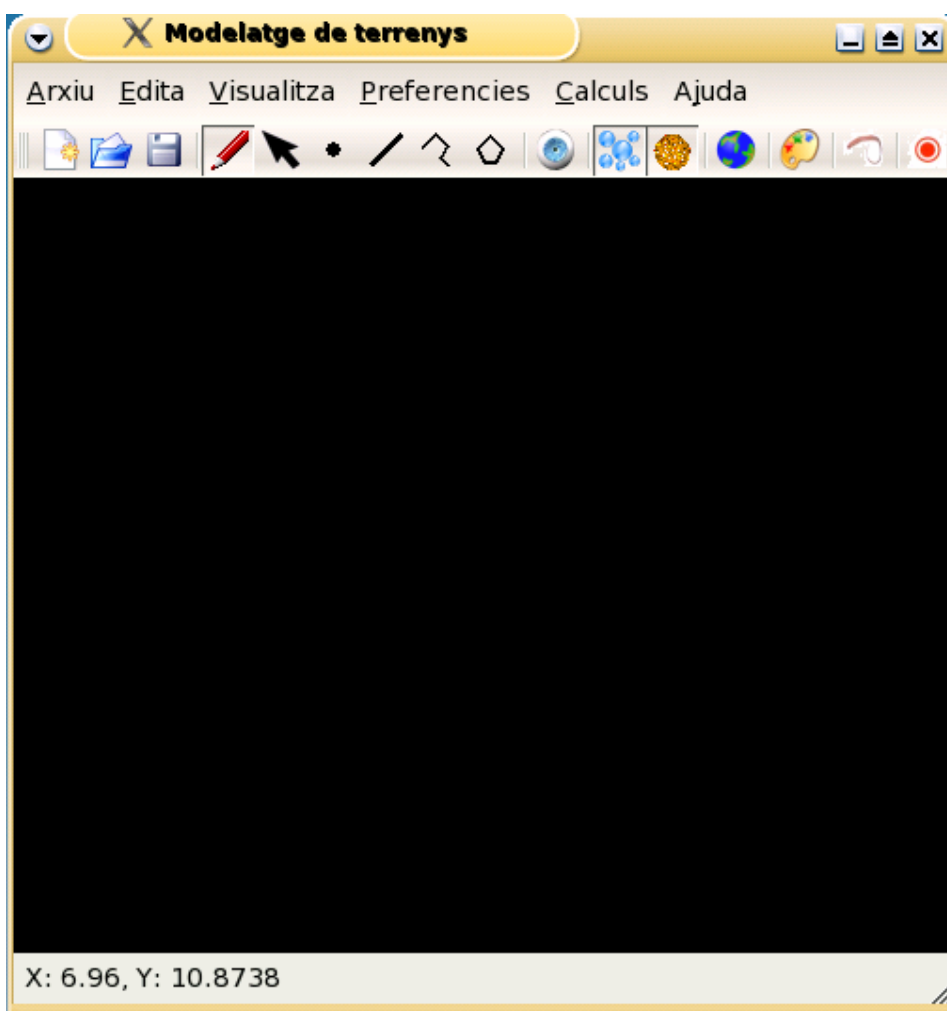


Figura 8.1 : Finestra principal o Editor 2D

Nota: Al parlar de opcions i menús que ens ofereix la finestra de l'Editor 2D només comentarem les opcions que s'utilitzen per tal de resoldre la problemàtica que afecta al nostre projecte. Les opcions que ja tenia la interfície de l'Editor 2D en la seva versió inicial i que no intervenen per res en el nostre projecte, no es tindran en compte.

Tal i com es pot observar en la Figura anterior, la interfície de l'Editor 2D disposa d'un gran nombre de botons i de menús, que permetran a l'usuari en tot moment realitzar totes les operacions que desitgi en l'Editor 2D. S'hi troben la majoria de funcionalitats de l'aplicació. Es troba dividit principalment en quatre regions:

- **Menú principal:** Les operacions es troben classificades per menús desplegable segons funcionalitats. En la figura següent podrem observar quins són els menús dels quals disposa l'Editor 2D.

Arxiu Edita Visualitza Preferències Càlculs Ajuda

Figura 8.2 : Menú principal de l'Editor 2D

Tot seguit es descriu el contingut de cadascun dels menús:

- **Arxiu** : Engloba les opcions bàsiques que permeten generar, emmagatzemar i carregar terrenys. A més a més, disposa de la opció *Exportar a imatge* que permet fer una captura de pantalla del terreny. Finalment, també ens permet sortir de l'aplicació.

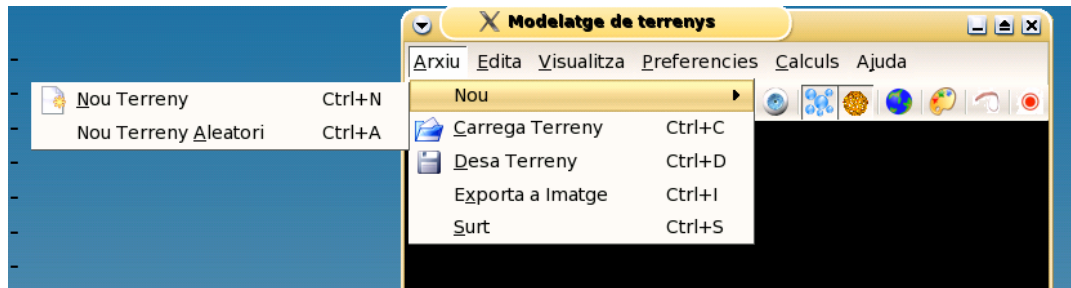


Figura 8.3 : Menú Arxiu de l'Editor 2D

- **Edita** : Permet escollir el tipus d'element restringit o seu que es vol inserir

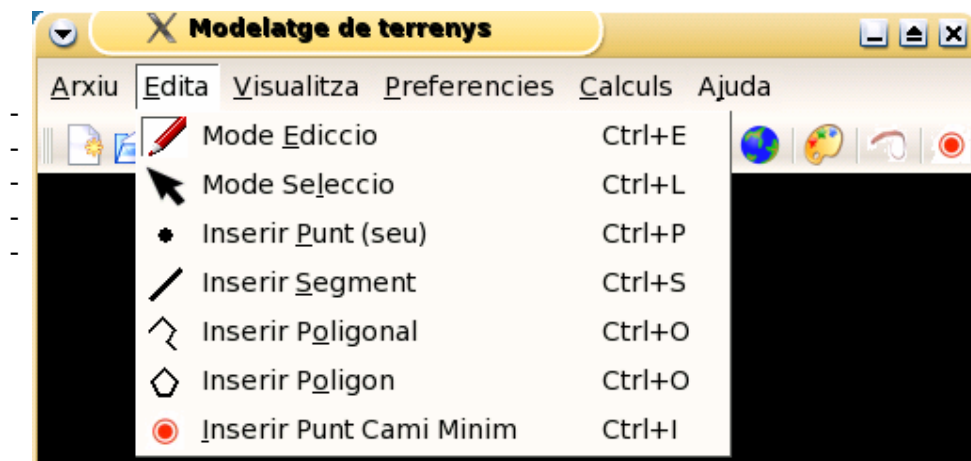


Figura 8.4 : Menú Editar de l'Editor 2D

- **Visualitza** : Podem trobar-hi la opció que permet amagar o mostrar la finestra del *Visor 3D*, així com les opcions per tal de veure les seus o elements restringits del terreny i els *Diagrames de Voronoi* per a un conjunt de seus



Figura 8.5 : Menú Visualitza de l'Editor 2D

- **Preferències** : Conté el submenú que ens permet accedir a les opcions de configuració de l'aplicació.



Figura 8.6 : Menú Preferències de l'Editor 2D

- **Càlculs** : Conté les opcions per decidir si volem treballar amb mode *Visibilitat* o mode *Distàncies*, així com les opcions de propagar les finestres al llarg del terreny i de trobar el camí més curt entre una o diverses seus i un o diversos punts del terreny

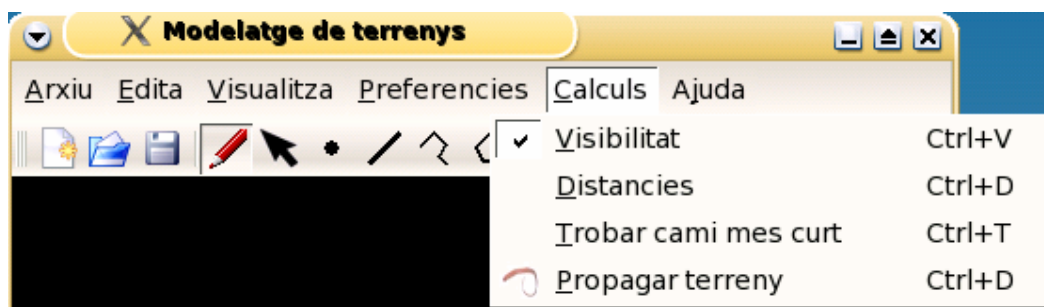


Figura 8.7 : Menú Càlculs de l'Editor 2D

- **Ajuda** : Disposa de la informació referent a l'aplicació.

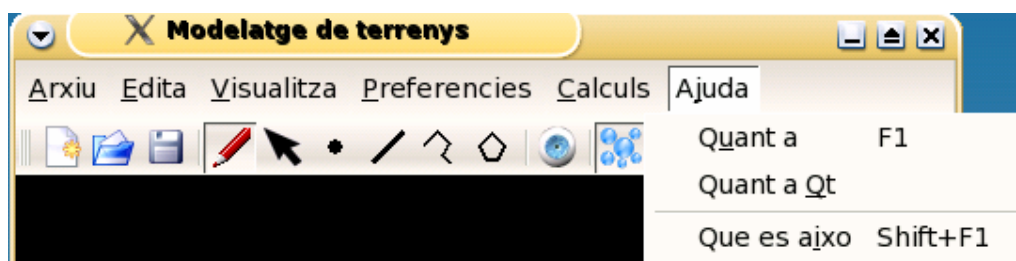


Figura 8.8 : Menú Ajuda de l'Editor 2D

- **Barra d'eines:** Conté les operacions més importants de l'aplicació. Representen una mena d'accés directe a les opcions del menú, ja que es troben lligades entre si. Qualsevol acció que es pugui fer a través de la barra d'eines també es podrà fer a través d'algun menú de la finestra de l'Editor 2D.



Figura 8.9 : Barra d'eines de l'Editor 2D

Les opcions de la barra d'eines es poden agrupar segons la classificació feta en el menú principal en:

- **Arxiu**
- **Edita**
- **Visualitza**
- **Càlculs**

- **Espai de treball:** Àrea de dibuix on serà editat el terreny. Serà a través d'aquesta regió amb la qual l'usuari haurà de realitzar gran part de la interacció.
- **Barra d'estats:** Indicarà la posició del ratolí (coordenades X i Y) sobre el terreny en tot moment.

8.2.2 Visor 3D

És la finestra que fa la funció de visualitzador del model del terreny en l'espai 3D. Podrà ésser o no visualitzada segons es seleccioni la opció corresponent de la anterior finestra principal, tal i com veurem més endavant. Des d'aquesta finestra, l'usuari serà capaç de veure el terreny des de diferents perspectives.

En la *Figura 8.10* de la pàgina següent podem observar quina és la forma que té la interfície final del programa pel que fa al *Visor 3D* un cop executem el programa.

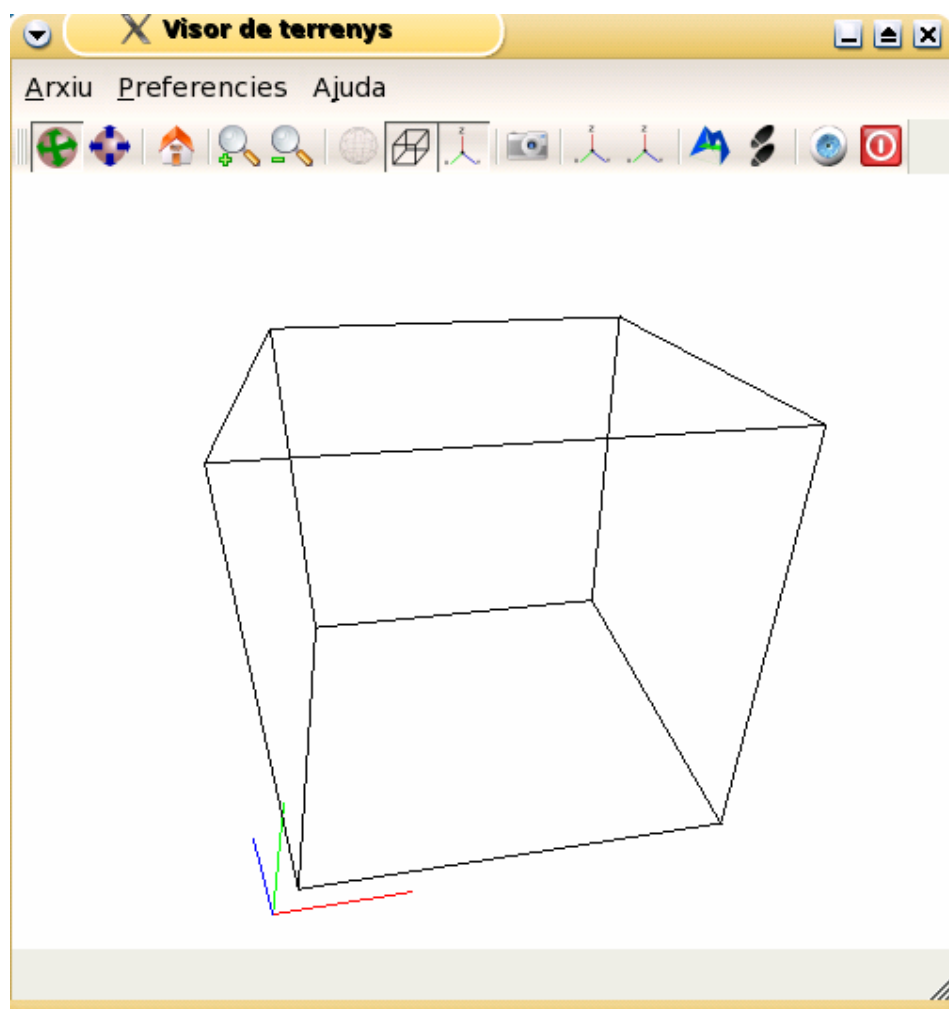


Figura 8.10 : Visualitzador o Visor 3D

Nota: Al parlar de opcions i menús que ens ofereix la finestra del Visor 3D només comentarem les opcions que s'utilitzen per tal de resoldre la problemàtica que afecta al nostre projecte. Les opcions que ja tenia la interfície del Visor 3D en la seva versió inicial i que no intervenen per res en el nostre projecte, no es tindran en compte.

Tal i com es pot observar en la *Figura 8.10*, la interfície del *Visor 3D* disposa d'una sèrie de botons i de menús, que permetran a l'usuari en tot moment realitzar totes les operacions que desitgi en el *Visor 3D*. La finestra del visualitzador o *Visor 3D* té una estructura semblant a la finestra principal (*Editor 2D*) però amb menys opcions:

- **Menú principal:** Les operacions es troben classificades per menús desplegable segons funcionalitats. En la figura següent podrem observar quins són els menús dels quals disposa el *Visor 3D*.

Arxiu Preferències Ajuda

Figura 8.11 : Menú principal del Visor 3D

Tot seguit es descriu el contingut de cadascun dels menús:

- **Arxiu** : Disposa de la opció *Captura imatge* que permet fer una captura de pantalla del terreny

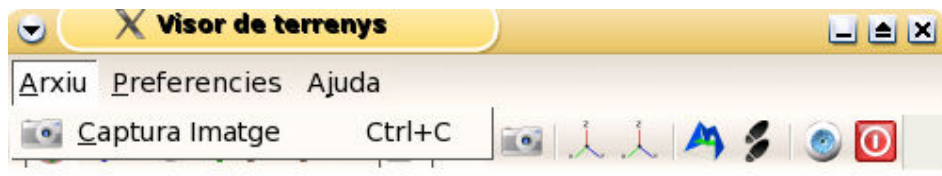


Figura 8.12 : Menú Arxiu del Visor 3D

- **Preferències** : Conté el submenú que ens permet accedir a les opcions de configuració de l'aplicació.

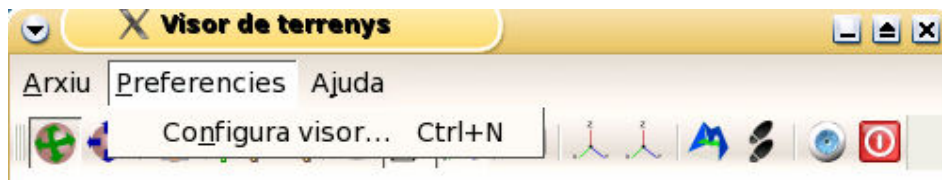


Figura 8.13 : Menú Preferències del Visor 3D

- **Ajuda** : Disposa de la informació referent a l'aplicació.

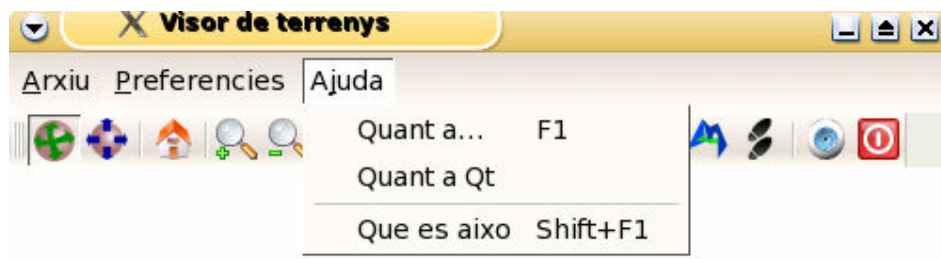


Figura 8.14 : Menú Ajuda del Visor 3D











- **Barra d'eines**: Conté totes les operacions que es permeten realitzar des del *Visor 3D*: situació de la càmera, opcions de visualització, veure seus inserides en el terreny, veure camins òptims entre seu i punt, veure finestres del terreny, ... A més a més, hi ha moltes icones que realitzen alguna funció a la qual només s'hi pot accedir via barra d'eines, és a dir, que és accessible via barra d'eines i no ho és via menú



Figura 8.15 : Barra d'eines del Visor 3D

Les opcions de la barra d'eines es poden agrupar segons la classificació feta en el menú principal en:

- **Funcionalitats de botons (que no es troben a cap menú)**

- *Rotar càmera* 
- *Desplaçar càmera* 
- *Iniciar vista (Home)* 
- *Apropar càmera (+)* 
- *Allunyar càmera (-)* 
- *Veure límits de l'espai* 
- *Veure eixos de coordenades* 
- *Finestres* 
- *Elements Visibles* 
- *Surt del programa* 

- **Espai de treball:** En aquest hi visualitzarem en 3D el terreny que tenim editat en la finestra principal. Podrem rotar, allunyar i apropar la càmera per tal de veure el terreny des de diferents punts de vista. Inicialment dins d'aquest espai de treball hi trobarem, en el centre, l'esquelet d'un cub amb uns eixos de coordenades indicant el punt (0,0,0). Per defecte, els eixos estan pintats un de cada color per tal de poder distingir l'eix X (color vermell), Y (color verd) i Z (color blau). D'aquesta manera, tindrem un punt de referència i ens podrem situar en qualsevol moment.

Cal dir que s'ha intentat mantenir la coherència pel que fa a les icones, ja que tant en les opcions dels menús com en els botons de les barres d'eines apareixen els mateixos dibuixos en les operacions equivalents. S'ha intentat que les icones representessin la funcionalitat que duïen a terme, utilitzant en la mesura del possible icones estàndard, com per exemple la *d'obrir fitxer* o *guardar fitxer*. A més a més de botons amb icones, també s'han utilitzat altres elements com poden ésser *radio buttons*, *checkbox*, *llistes desplegables*, ...

8.3 Opcions del programa

Després d'haver observat quin és l'aspecte de les dues interfícies que componen l'aplicació, parlarem sobre el funcionament de cadascuna de les funcions més importants de què disposa l'aplicació. Han estat implementades totes les funcionalitats sol·licitades durant la fase de requeriments. A continuació, doncs, explicarem amb detall per a què serveix cada opció de menú o botó, tant de l'*Editor 2D* com del *Visor 3D*.

Nota: *Al parlar de les opcions i menús que ens ofereix l'aplicació (tant de l'Editor 2D com del Visor 3D) només comentarem les opcions que s'utilitzen per tal de resoldre la problemàtica que afecta al nostre projecte. Les opcions que ja tenia la interfície de l'aplicació en la seva versió inicial i que no intervenen per res en el nostre projecte, no s'explicaran en aquest manual.*


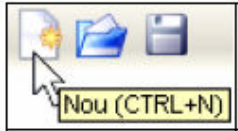
Les funcionalitats de l'aplicació que explicarem són les següents:

- *Crear Nou terreny*
- *Crear Nou Terreny Aleatori*
- *Carregar un terreny*
- *Desar un terreny*
- *Guardar informació en format gràfic*
- *Inserir seus en el terreny*
- *Inserir elements restringits en el terreny*
- *Inserir Punt Camí Mínim en el terreny*
- *Obtenir informació*
- *Modificar el terreny*
 - *Modificar alçada de punts del terreny*
 - *Eliminar Elements*
- *Modificar configuració Editor 2D*
- *Modes de treball (Visibilitat / Distàncies)*
- *Propagar terreny*
- *Trobar camí més curt*

- *Visualitzar elements visibles*
- *Visualitzar Diagrames de Voronoi*
- *Ajuda*
- *Sortir de l'aplicació*
- *Mostrar / Amagar Visor 3D*
 - *Opcions de navegació: moviment de la càmera*
 - *Realitzar zoom*
 - *Opcions de visualització*
 - *Guardar informació en format gràfic*
 - *Modificar configuració del Visor 3D*
 - *Veure finestres*
 - *Visualitzar elements visibles*
 - *Ajuda*
 - *Tancar el Visor 3D*

8.3.1 Crear nou Terreny

En el moment que l'usuari inicia l'aplicació disposa d'un nou terreny buit per tal d'ésser editat. Aquesta possibilitat també pot ésser escollida en qualsevol moment en què l'usuari interactuï amb l'aplicació, és a dir, l'usuari podrà crear un nou terreny en el moment en què ho desitgi. Per tal de dur a terme aquesta operació, disposarem de tres maneres diferents de fer-ho:

- Des del menú principal, accedint a la opció **Nou → Nou Terreny** del menú **Arxiu**.

- Prement el botó *Nou Terreny* de la barra d'eines de la finestra principal

- Utilitzant la combinació de tecles *CTRL + N* tal i com es mostra al costat de l'operació *Nou Terreny* de dins del menú *Arxiu*.

En qualsevol dels casos anteriors, *l'Editor 2D* i el *Visor 3D* (en cas que es trobi activat), quedaran buits i preparats per tal de poder-hi introduir les dades d'un nou terreny.

Per defecte, els límits del terreny seran de **0.0 – 20.0** per a les tres coordenades X, Y i Z. En el cas que es desitgin modificar les mides superiors i/o inferiors del terreny, caldrà modificar la configuració de *l'Editor 2D* tal i com s'explica més endavant en l'apartat *Modificar Configuració Editor 2D*. En aquests moments, ja estaríem en condicions de poder generar un nou terreny, ja sigui dibuixat per nosaltres (inserir punts en el terreny i triangulant el núvol de punts), generar un terreny aleatori (apartat *Crear Nou Terreny aleatori*) o carregar un terreny des d'un fitxer (apartat *Carregar un Terreny*)

8.3.2 Crear Nou Terreny Aleatori

Un cop tinguem un terreny en blanc, estarem en condicions de generar un terreny aleatori. Per fer-ho, l'usuari tan sols haurà d'accedir a la opció **Nou → Nou Terreny Aleatori** del menú **Arxiu**, o bé, a través de la combinació de les tecles **CTRL + A**.

Nou Terreny Aleatori Ctrl+A

Un cop seleccionada la opció, apareixerà una finestra de diàleg a través de la qual se'ns demanarà introduir un valor enter que indicarà el nombre d'iteracions que es vol que faci l'algorisme generador de terrenys aleatoris, fins a un màxim de 10000 iteracions. (**Nota:** *L'algorisme implementat es basa en la descomposició de partícules i utilitza aquest paràmetre en el càlcul de les alçades del punt*)



El resultat serà un terreny generat a partir d'una malla rectangular, creada segons les mides del terreny indicades en la configuració, on cadascun dels vèrtexs tindrà un valor d'alçada aleatori. Aquest podrà ésser visualitzar en *l'Editor 2D* i en el *Visor 3D* en el cas que aquest últim estigui activat.

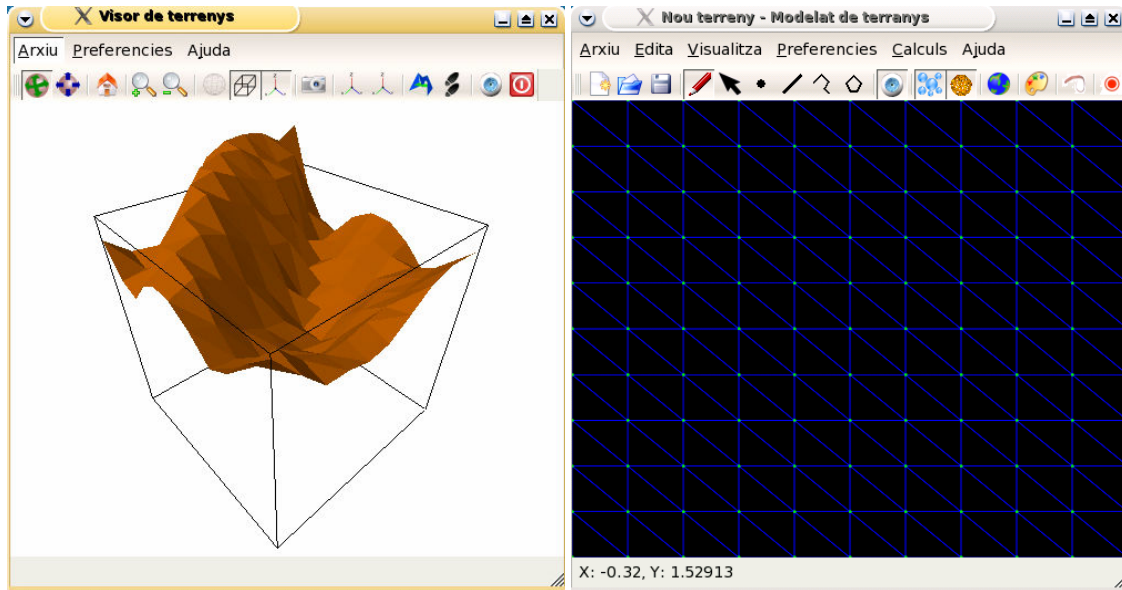


Figura 8.16 : Terreny generat de forma aleatòria

8.3.3 Carregar un terreny

Una altra possibilitat per tal de generar terrenys consisteix en carregar les dades des d'un fitxer. Caldrà, però, que el format del fitxer sigui algun dels suportats per l'aplicació. Per tal de poder realitzar la càrrega d'un terreny emmagatzemat en un fitxer tindrem diverses possibilitats:

- Des del menú principal, accedint a la opció **Carrega Terreny** del menú **Arxiu**.



- Prémer el botó **Carrega Terreny** de la barra d'eines de la finestra principal



- Utilitzar la combinació de tecles **CTRL + C**, tal i com es mostra al costat de l'opció **Carrega Terreny** del menú **Arxiu**

Un cop seleccionada la operació utilitzant una de les opcions anteriors, apareixerà la típica finestra de diàleg d'obrir fitxer, la qual ens demanarà el directori i el nom del fitxer on es troba el terreny que es desitja carregar.

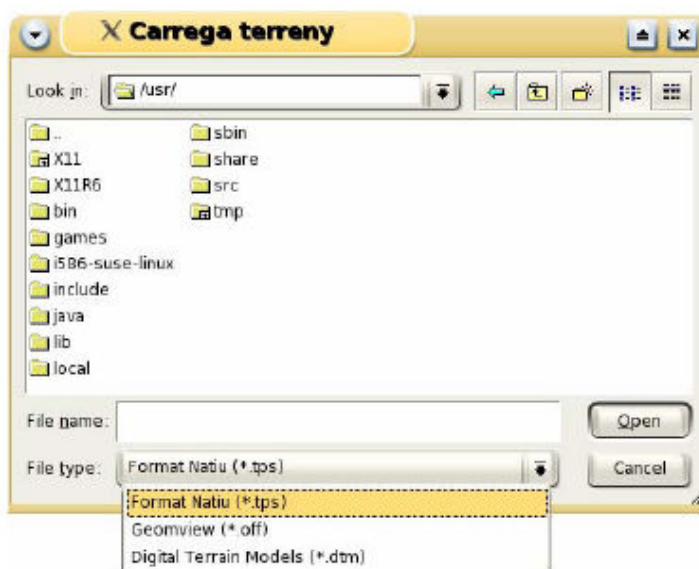


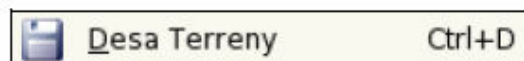
Figura 8.17 : Finestra de diàleg per tal de carregar un terreny amb algun dels formats suportats per l'aplicació

Un cop seleccionat el fitxer, es comprovarà si el format del contingut és correcte i, a més a més, és un dels formats suportats per l'aplicació explicats en *l'apartat 5.2* d'aquesta memòria. En cas que existeixi algun error durant la càrrega, s'informaria a l'usuari a través d'un missatge d'error. Com a resultat de què la carrega del fitxer ha estat correcte, serà representat el terreny llegit tant en *l'Editor 2D* com en el *Visor 3D*, en el cas que aquest es trobi actiu.

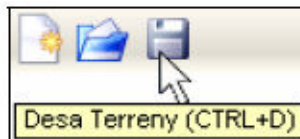
8.3.4 Desar un terreny

Qualsevol terreny creat o editat a través de l'aplicació, ja sigui carregat des de fitxer i modificat posteriorment o creat des de zero, podrà ésser guardat en qualsevol dels formats suportats per l'aplicació. D'aquesta manera, podrà ésser carregat de nou en qualsevol moment. Per tal de poder desar un terreny en un fitxer disposem de diverses formes de fer-ho:

- Des del menú principal accedint a la opció **Desa Terreny** del menú **Arxiu**.



- Prement el botó *Desa Terreny* de la barra d'eines de la finestra principal.



- Utilitzant la combinació de tecles *CTRL + D*, tal i com es mostra al costat de l'opció *Desa Terreny* del menú *Arxiu*

Un cop escollia alguna de les opcions anteriors, se'ns mostrarà la finestra de diàleg de la *Figura 8.18*, a través de la qual caldrà indicar el directori i el nom del fitxer on es voldrà guardar. A més a més, caldrà seleccionar l'extensió amb que volem que quedi guardat segons les diferents possibilitats de formats que apareixen en la llista desplegable *File type*.

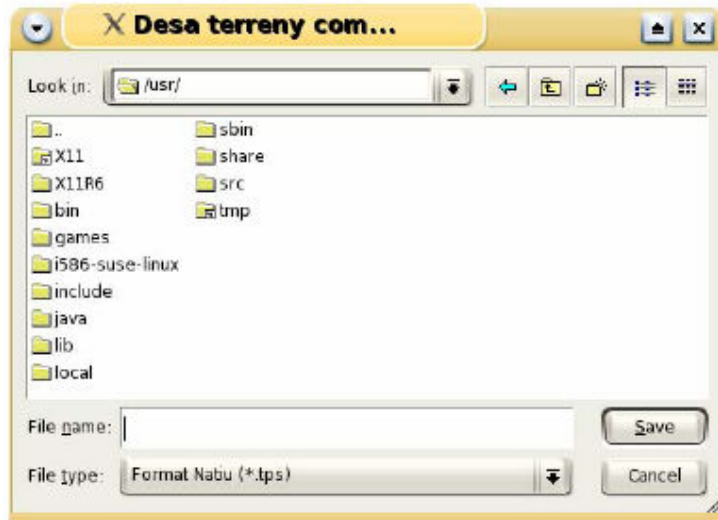


Figura 8.18 : Finestra de diàleg per tal de desar un terreny

8.3.5 Guardar informació en format gràfic

Com a funcionalitat afegida, l'aplicació permet realitzar una captura d'imatge de l'espai de treball de l'**Editor 2D** en format gràfic. Podrem accedir a aquesta opció a través d'alguna de les següents formes:

- Des del menú principal, accedint a la opció **Exporta a imatge** del menú **Arxiu**

Exporta a Imatge Ctrl+I
- Utilitzant la combinació de tecles *CTRL + I*, tal i com es mostra al costat de l'opció *Exporta a imatge* del menú *Arxiu*

Un cop seleccionada la operació apareixerà una finestra de diàleg a través de la qual caldrà indicar el nom del fitxer amb el qual es vol desar la imatge. A més a més, haurem d'escollir un dels formats d'imatges suportats:

Format fitxers d'imatge							
.bmp	.jpg	.pbm	.png	.xpm	.xbm	.ppm	.pgm

El resultat serà un fitxer amb el nom i formats indicats que contindrà la imatge corresponent al terreny visible des de l'**Editor 2D**. D'aquesta manera, podrem visualitzar-la posteriorment amb qualsevol editor d'imatges.

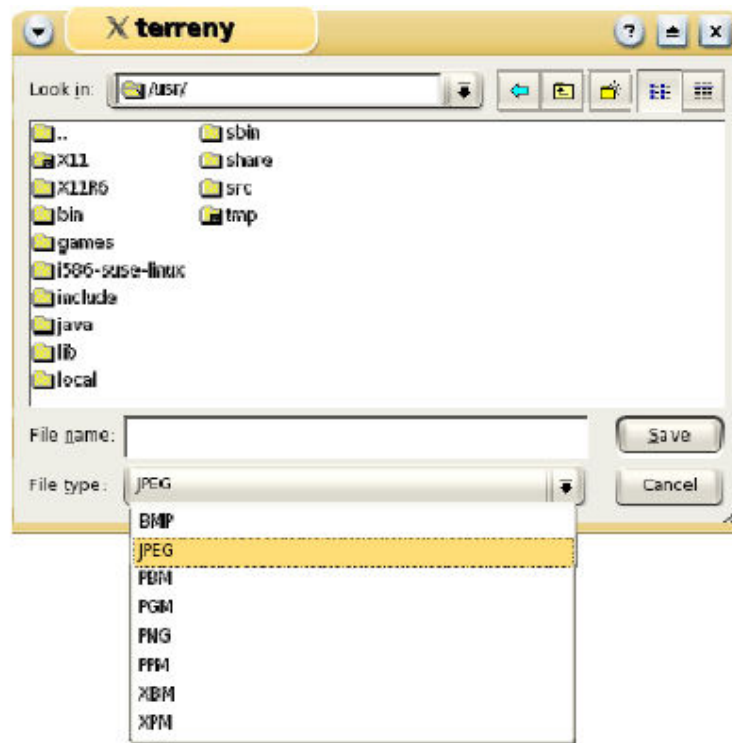


Figura 8.19 : Finestra de diàleg per tal de guardar com a imatge el terreny

8.3.6 Inserir seus en el terreny

Una de les principals funcionalitats de l'aplicació consisteix en permetre inserir seus (elements de distàncies) dins del terreny. A través de la finestra principal tenim la possibilitat d'introduir punts dins del terreny que actuïn com a seu (més endavant està pensat introduir la possibilitat de poder inserir segments, polígons i poligonals que actuïn com a seu). A continuació anem a descriure els passos que cal seguir per tal de poder inserir punts que actuïn com a seu en el terreny. (Suposarem que ja tenim el terreny generat a la pantalla)

- Des del menú principal activar la modalitat de treball **Distàncies** del menú **Calculs**, o bé prement la combinació de tecles **CTRL + D**.

<u>D</u> istàncies	Ctrl+D
--------------------	--------
- Des del menú principal accedir a la opció **Inserir punt (seu)** del menú **Edita**, o prement el botó *Inserir Punt (seu)* de la barra d'eines de la finestra principal, o prement la combinació de tecles **CTRL + P**.

✱ Inserir <u>P</u> unt (seu)	Ctrl+P
------------------------------	--------

Un cop seguits aquests dos passos, ja estem preparats per tal d'inserir punts en el terreny que es convertiran en seus. Per fer-ho, només caldrà clicar amb el botó esquerre del ratolí sobre el punt del terreny on desitgem introduir-hi la nova seu. Per tal de poder determinar el punt amb coordenades exactes podem ajudar-nos de la barra d'estats a través de la qual veurem les coordenades (X,Y) reals de la posició del ratolí sobre el terreny. En el moment que deixem anar el botó, ja tindrem el punt (seu) inserit en el terreny. La coordenada Z (alçada del punt) no la introduïm nosaltres, sinó que ja ens la calcula el propi programa, posicionant-lo a sobre de la cara que el conté.

Tal i com ja hem comentat, i com es pot observar en la *Figura 8.20*, la opció de inserir segments, polígons o poligonals que actuïn com a seus de moment encara no està operativa

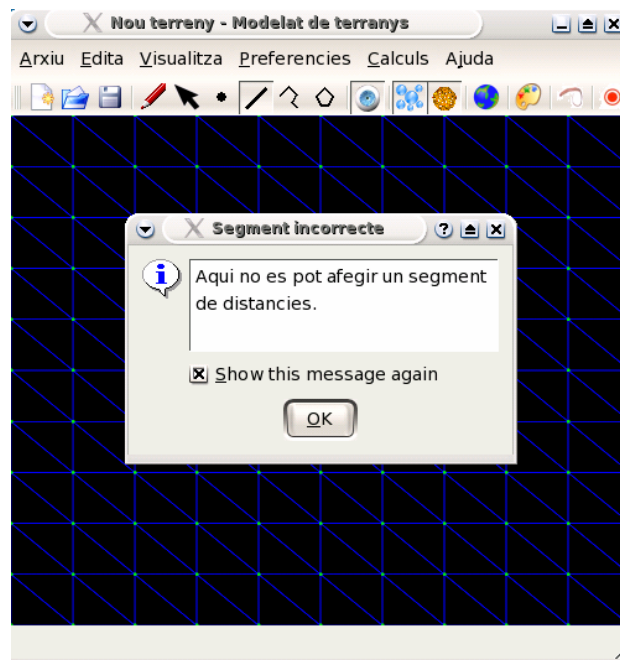


Figura 8.20 : Missatge d'error al intentar inserir un segment en el terreny que actuï com a seu.

8.3.7 Inserir elements restringits en el terreny

Una altra de les funcionalitats de l'aplicació consisteix en permetre inserir elements restringits dins del terreny. A través de la finestra principal tenim la possibilitat d'introduir: punts, segments, polígons i poligonals dins del terreny. Un cop inserits cadascun d'ells, faran que la malla del terreny quedi correctament triangulada afegint-los dins de la triangulació. Tot seguit es descriuen els passos a seguir per tal de poder inserir cadascun dels diferents tipus d'elements restringits dins del terreny.

→ **Punt:**

- Des del menú principal activar la modalitat de treball **Visibilitat** del menú **Calculs**, o bé prement la combinació de tecles **CTRL + V**.

	✓	<u>V</u> isibilitat	Ctrl+V
--	---	---------------------	--------
- Des del menú principal accedir a la opció **Inserir punt (seu)** del menú **Edita**, o prement el botó *Inserir Punt (seu)* de la barra d'eines de la finestra principal, o prement la combinació de tecles **CTRL + P**.

✱	Inserir <u>P</u> unt (seu)	Ctrl+P
---	----------------------------	--------

Un cop seguits aquests dos passos, ja estem preparats per tal d'inserir punts en el terreny que es convertiran en vèrtexs restringits dins de la triangulació. Per fer-ho, només caldrà clicar amb el botó esquerre del ratolí sobre el punt del terreny on desitgem introduir-hi un nou vèrtex. Per tal de poder determinar el punt amb coordenades exactes podem ajudar-nos de la barra d'estats a través de la qual veurem les coordenades (X,Y) reals de la posició del ratolí sobre el terreny. En el moment que deixem anar el botó, ens apareixerà una finestra de diàleg com la de la *Figura 8.21* que ens determinarà la coordenada Z (alçada) del punt inserit. El valor d'alçada podrà ésser introduït directament des de teclat, o bé, a través de la interacció del ratolí amb l'*slider* (barra lliscant). Després d'això, ja hurem aconseguit introduir el punt dins del terreny amb les coordenades desitjades.



Figura 8.21 : Finestra de diàleg per tal d'introduir l'alçada del punt

→ **Poligonal:**

- Des del menú principal activar la modalitat de treball **Visibilitat** del menú **Calculs**, o bé prement la combinació de tecles **CTRL + V**.

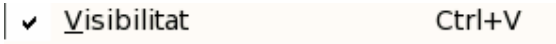
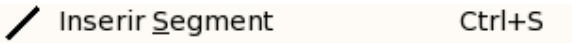
	✓	<u>V</u> isibilitat	Ctrl+V
--	---	---------------------	--------
- Des del menú principal accedir a la opció **Inserir poligonal** del menú **Edita**, o bé prement el botó *Inserir Poligonal* de la barra d'eines de la finestra principal, o bé prement la combinació de tecles **CTRL + O**.

↗	Inserir <u>P</u> oligonal	Ctrl+O
---	---------------------------	--------

Un cop seguits aquests dos passos, ja estem disposats a introduir la poligonal dins del terreny. En aquest cas, l'usuari haurà d'anar introduint punts tal i com s'ha explicat en el cas anterior. Els diversos punts s'aniran unint a

través de segments. L'últim punt introduït fins al moment serà unit amb la posició del ratolí, indicant així com quedaria el segment si inserís aquell punt. Per tal de finalitzar la poligonal, l'últim punt a inserir haurà d'ésser clicant amb el botó dret del ratolí en comptes de fer-ho amb el botó esquerre. Com a resultat podrem veure la malla triangulada amb la nova poligonal inserida.


→ Segment:

- Des del menú principal activar la modalitat de treball **Visibilitat** del menú **Calculs**, o bé prement la combinació de tecles **CTRL + V**.

- Des del menú principal accedir a la opció **Inserir segment** del menú **Edita**, o bé prement el botó *Inserir Segment* de la barra d'eines de la finestra principal, o bé prement la combinació de tecles **CTRL + S**.


Un cop seguits aquests dos passos, ja estem preparats per tal d'inserir segments en el terreny. Per fer-ho, l'usuari haurà d'introduir dos punts, un d'inici de segment i l'altra de fi de segment, que s'uniran i definiran el nou segment. Després d'això, ja hurem aconseguit introduir el segment dins del terreny amb les coordenades desitjades.

Una altra manera de poder inserir segments en el terreny és a través de la opció *Inserir Poligonal*, considerant un segment com a una poligonal amb un únic segment. Així doncs, llavors tan sols haurà d'afegir els dos punts que defineixen el segment .


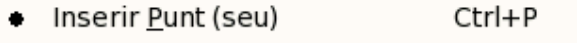
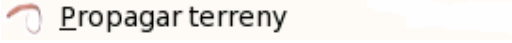
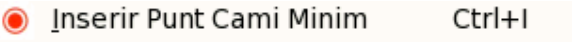
→ Poligon:

- Des del menú principal activar la modalitat de treball **Visibilitat** del menú **Calculs**, o bé prement la combinació de tecles **CTRL + V**.

- Des del menú principal accedir a la opció **Inserir polígon** del menú **Edita**, o bé prement el botó *Inserir Polígon* de la barra d'eines de la finestra principal, o bé prement la combinació de tecles **CTRL + G**.

Un cop seguits aquests dos passos, la introducció del polígon dins del terreny és molt senzilla. Tan sols caldrà realitzar el mateix procés que en el cas d'inserir Poligonal vist anteriorment. La única diferència es troba en el fet que l'usuari podrà anar veient el polígon resultant com si cada punt a inserir fos l'últim. D'aquesta manera, el primer punt introduït serà unit amb la posició del ratolí en tot moment. Un cop finalitzada la introducció del polígon, podrem veure el terreny triangulat amb el nou element.

8.3.8 Inserir punt camí mínim en el terreny

Una funcionalitat de l'aplicació que serà important a l'hora d'obtenir el camí mínim entre una seu i un punt, és la de poder inserir un punt de camí mínim en el terreny, és a dir, un punt que no actuï ni com a seu ni com a element restringit (vèrtex de la triangulació). A continuació anem a descriure els passos que cal seguir per tal de poder inserir punts de camí mínim en el terreny. (Suposarem que ja tenim el terreny generat a la pantalla)

- Des del menú principal activar la modalitat de treball **Distàncies** del menú **Càlculs**, o bé prement la combinació de tecles **CTRL + D**.

- Des del menú principal accedir a la opció **Inserir punt (seu)** del menú **Edita**, o bé prement el botó *Inserir Punt (seu)* de la barra d'eines de la finestra principal, o bé prement la combinació de tecles **CTRL + P**.

- Des del menú principal accedir a la opció **Propagar Terreny** del menú **Càlculs**, o bé prement el botó *Propagar Terreny* de la barra d'eines de la finestra principal

- Des del menú principal accedir a la opció **Inserir Punt Camí Mínim** del menú **Edita**, o bé prement el botó *Inserir Punt Camí Mínim* de la barra d'eines de la finestra principal, o bé prement la combinació de tecles **CTRL + I**.


Un cop seguits aquests passos, ja haurem inserit un punt de camí mínim en el terreny (que el visualitzarem de color groc, un color diferent al color de les seus i els vèrtexs de la triangulació), i ja estarem preparats per tal de poder calcular el camí més curt entre aquest punt que acabem d'inserir i la seu més propera. El nou punt de camí mínim inserit (de color groc), es pot visualitzar a la *Figura 8.22* de la pàgina següent.

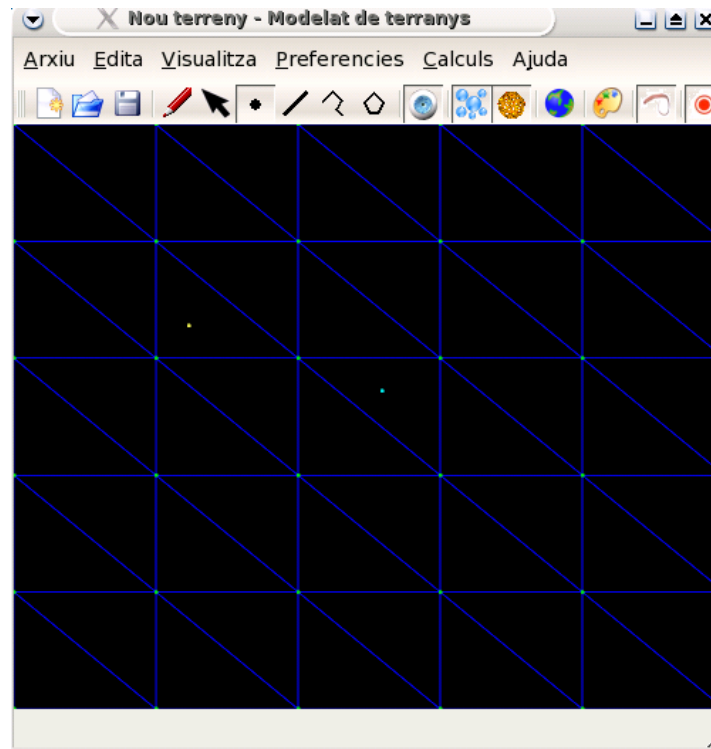
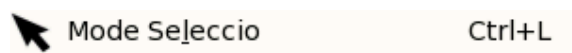


Figura 8.22 : Exemple d'inserció punt camí mínim (color groc) en el terreny

8.3.9 Obtenir informació

Una funcionalitat que ens ofereix la interfície és la de saber les coordenades exactes d'un vèrtex del terreny. Per poder saber aquesta informació, s'haurà de seleccionar el vèrtex del qual se'n vol obtenir informació. Per poder-lo seleccionar, s'ha de tenir seleccionat el **Mode Selecció** del menú **Edita** (també es pot utilitzar el botó *Mode Selecció* de la barra d'eines de la finestra principal o bé prement la combinació de tecles *CTRL + L*)



Un cop tenim el vèrtex seleccionat (ho realitzem a través d'un clic amb el botó esquerre del ratolí damunt del vèrtex), veurem com aquest canvia de color. A continuació, prement amb el botó dret del ratolí, seleccionarem la opció *Informació*, i ens apareixerà una finestra de diàleg que contindrà el valor de cadascuna de les coordenades X, Y i Z equivalents al vèrtex.

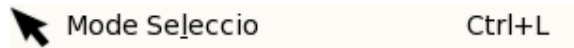


Figura 8.23 : Finestra de diàleg que mostra la informació (coordenades X, Y i Z) del punt seleccionat

8.3.10 Modificar el terreny

8.3.10.1 Modificar alçada de punts del terreny

Una de les possibilitats que ens ofereix l'aplicació consisteix en canviar l'alçada (coordenada Z) d'un punt escollit per l'usuari. Per poder modificar l'alçada, s'haurà de seleccionar el vèrtex en qüestió. Per poder-lo seleccionar, s'ha de tenir seleccionat el **Mode Selecció** del menú **Edita** (també es pot utilitzar el botó *Mode Selecció* de la barra d'eines de la finestra principal o bé prémer la combinació de tecles **CTRL + L**)



Un cop tenim el vèrtex seleccionat (ho realitzem a través d'un clic amb el botó esquerre del ratolí damunt del vèrtex), veurem com aquest canvia de color. A continuació, prement amb el botó dret del ratolí, seleccionarem la opció *Modifica Altura*, i ens apareixerà una finestra com la de la *Figura 8.21* que ens permetrà canviar l'alçada (coordenada Z) del vèrtex seleccionat. Si seleccionéssim més d'un punt i els hi volguéssim canviar l'alçada, tots agafarien la nova alçada configurada a través de la finestra de diàleg. En acceptar l'operació, podrem veure, a través del *Visor 3D* (en cas que es trobi activat), com els punts seleccionats hauran modificat la seva alçada amb el valor indicat per l'usuari.

8.3.10.2 Eliminar elements

Una altra possibilitat que ens ofereix el programari consisteix en poder eliminar elements. Actualment, parlarem de dos tipus possibles d'eliminacions: *eliminar punts (vèrtexs) restringits* (que formin part de la malla), i *eliminar punts que actuïn com a seu*. En tots els casos, el procediment a seguir serà el mateix. Per poder eliminar un element (ja sigui un vèrtex restringit o una seu), primerament s'haurà de seleccionar el punt o vèrtex en qüestió. Per poder-lo seleccionar, s'ha de tenir seleccionat el **Mode Selecció** del menú **Edita** (també es pot utilitzar el botó *Mode Selecció* de la barra d'eines de la finestra principal o bé prémer la combinació de tecles **CTRL + L**)

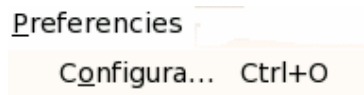


Un cop tenim el vèrtex o seu seleccionada (ho realitzem a través d'un clic amb el botó esquerre del ratolí damunt del vèrtex), veurem com aquest canvia de color. A continuació, prement amb el botó dret del ratolí, seleccionarem la opció *Elimina Element*, fet que farà que ens desaparegui l'element en qüestió.

En cas que eliminem un vèrtex de la triangulació, aquesta serà retriangulada de nou, i en el cas que eliminem un punt que actuï com a seu, aquest serà esborrat, fet que també es veurà reflectit immediatament en el *Visor 3D* en cas que el tinguem activat.

8.3.11 Modificar configuració Editor 2D

Una altra de les possibilitats que ens ofereix el programa és poder modificar la configuració de *l'Editor 2D*. Per poder accedir a la configuració de l'Editor 2D anirem al menú **Preferències** i seleccionarem l'opció **Configuració de l'editor**, o premerem la combinació de tecles **CTRL + O**



En aquest cas, ens apareixerà una finestra formada per quatre pestanyes, on cadascuna d'elles ens permetrà modificar aspectes diferents de la finestra:

- **Espai:** És la pestanya que es troba activada per defecte. Permet modificar les mides del terreny a partir de les coordenades X, Y i Z mínimes i màximes



Figura 8.24 : Finestra de configuració de les dimensions de l'espai a l'Editor 2D

L'usuari podrà introduir els valors que desitgi (enters o reals) i haurà de prémer el botó **Accepta**. La modificació de la mida del terreny afectarà en l'espai de treball visualitzat, i a més a més, en el cas de generar un terreny aleatori la mida del grid s'adaptarà a la mida configurada.

- **Colors:** La segona pestanya de la finestra permet modificar el color dels elements bàsics de *l'Editor 2D*, com són:



Figura 8.25 : Finestra de configuració dels colors a l'Editor 2D

- **Color de Fons:** Color de fons de l'espai de treball de l'Editor 2D
- **Color Línies:** Color de les arestes de la triangulació
- **Color Punts:** Color amb què seran representats els vèrtexs restringits de la triangulació
- **Color de Selecció:** Color utilitzat en la selecció d'elements

En tots els casos, la selecció del color es fa a través de la mateixa finestra de colors. Un cop introduïts els colors que volem per cada element, tant sols ens caldrà acceptar la operació. En aquest cas podrem veure com els canvis de color es duen a terme de forma instantània. Comentar també el fet que en qualsevol moment, l'usuari podrà cancel·lar l'operació, prement el botó **Cancel·la**.

- **Graella i Discretització:** Les dues últimes finestres de la finestra (*Graella* i *Discretització*) no les utilitzarem en el nostre programa, ja que les opcions que ofereixen no intervien dins la nostra problemàtica.

8.3.12 Modes de treball (Visibilitat / Distàncies)

Un aspecte molt important a l'hora d'interactuar amb la nostra aplicació és saber en quin mode de treball volem treballar. Diferenciarem dos modes de treball diferents: **Visibilitat** i **Distàncies**. El que utilitzarem més a l'hora de resoldre els problemes de proximitat en terrenys i per tal de poder inserir seus

en el terreny, és el mode *Distàncies*, que és el que es troba actiu per defecte en iniciar el programa. El mode *Visibilitat* gairebé només l'utilitzarem per tal d'inserir elements restringits (punts, segments, polígons i poligonals) en el terreny.

Per tal de canviar de mode de treball, hem d'anar al menú **Càlculs** i activar el mode de treball que desitgem (**Visibilitat** o **Distàncies**). Com es pot suposar, els dos modes són complementaris, és a dir, que quan en marquem un, automàticament l'altra quedarà desmarcat.

✓ <u>V</u> isibilitat	Ctrl+V
<u>D</u> istàncies	Ctrl+D

Per tal de seleccionar el mode *Visibilitat* també es pot utilitzar la combinació de tecles **CTRL + V**, i per tal de seleccionar el mode *Distàncies* es pot utilitzar la combinació de tecles **CTRL + D**.


8.3.13 Propagar terreny

Una de les funcionalitats més utilitzades per tal de poder obtenir el camí *mínim entre una seu i un punt* o per tal de poder veure el *Diagrama de Voronoi* per a un conjunt de seus, serà la de **Propagar terreny**, que consistirà, tenint un terreny amb una o diverses seus inserides, en anar posant finestres i anar-les propagant al llarg de tot el terreny. Per tal de poder realitzar aquesta opció, hauréu de seguir els següents passos:

- Des del menú principal activar la modalitat de treball **Distàncies** del menú **Càlculs**, o bé prémer la combinació de tecles **CTRL + D**.

<u>D</u> istàncies	Ctrl+D
--------------------	--------
- Des del menú principal accedir a la opció **Inserir punt (seu)** del menú **Edita**, o bé prémer el botó *Inserir Punt (seu)* de la barra d'eines de la finestra principal, o bé prémer la combinació de tecles **CTRL + P**.

✱ Inserir <u>P</u> unt (seu)	Ctrl+P
------------------------------	--------
- Des del menú principal accedir a la opció **Propagar Terreny** del menú **Càlculs**, o bé prémer el botó *Propagar Terreny* de la barra d'eines de la finestra principal

 **Propagar terreny**

Una vegada haguem realitzat aquests passos, veurem com es van creant finestres i propagant al llarg de tot el terreny, partint de la seu o del conjunt de seus que hem inserit prèviament en el terreny. Cada seu pintarà les finestres que surten d'ella d'un color diferent, tal i com es pot observar en la *Figura 8.26*. Una vegada haguem propagat totes les finestres al llarg del terreny, ja estarem en condicions de resoldre el problema de trobar el *camí més curt* entre una seu i un punt, o bé de visualitzar el *Diagrama de Voronoi* per a aquell terreny.

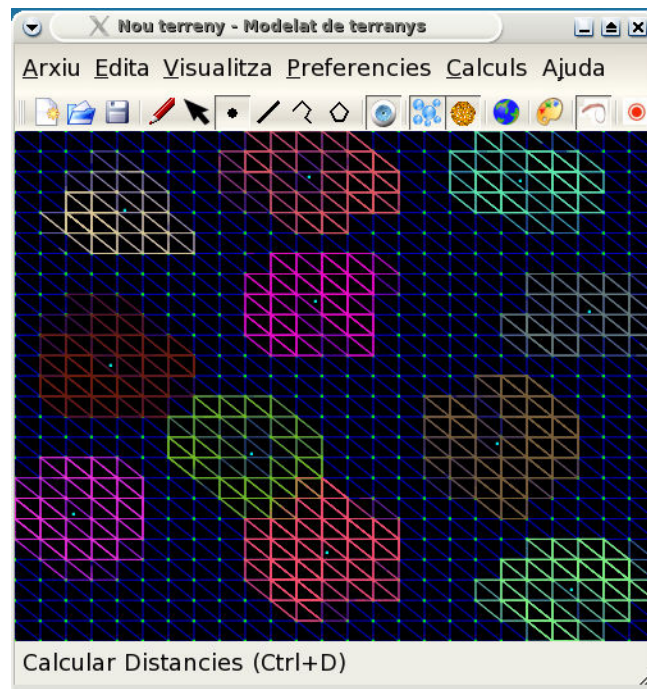



Figura 8.26 : Exemple de propagació de finestres per a un conjunt de seus


8.3.14 Trobar camí més curt


Una altra de les funcionalitats més utilitzades serà la d'obtenir el *camí més curt entre una seu i un punt*. Aquesta funcionalitat consistirà, tenint un terreny ja propagat amb totes les finestres, en reconstruir el camí mínim per anar d'un punt de camí mínim inserit en el terreny a la seu més propera que tingui. Per tal de poder realitzar aquesta opció, haurem de seguir els següents passos:

- Des del menú principal activar la modalitat de treball **Distàncies** del menú **Calculs**, o bé prémer la combinació de tecles **CTRL + D**.

Distàncies	Ctrl+D
-------------------	--------
- Des del menú principal accedir a la opció **Inserir punt (seu)** del menú **Edita**, o bé prémer el botó *Inserir Punt (seu)* de la barra d'eines de la finestra principal, o bé prémer la combinació de tecles **CTRL + P**.

 Inserir Punt (seu)	Ctrl+P
---	--------
- Des del menú principal accedir a la opció **Propagar Terreny** del menú **Calculs**, o bé prémer el botó *Propagar Terreny* de la barra d'eines de la finestra principal

 Propagar terreny	
---	--
- Des del menú principal accedir a la opció **Inserir Punt Camí Mínim** del menú **Edita**, o bé prémer el botó *Inserir Punt Camí Mínim* de la barra d'eines de la finestra principal, o bé prémer la combinació de tecles **CTRL + I**.

 Inserir Punt Camí Mínim	Ctrl+I
--	--------

- Des del menú principal accedir a la opció **Trobar camí més curt** del menú **Càlculs**, o bé prémer la combinació de tecles **CTRL + T**

Trobar camí mes curt Ctrl+T

Una vegada haguem realitzat aquests passos, veurem com es reconstrueix el camí mínim entre el punt inserit i la seu que tingui més propera. Aquest camí es visualitzarà tant en l'*Editor 2D* com en el *Visor 3D*, en el cas que aquest últim estigui activat.

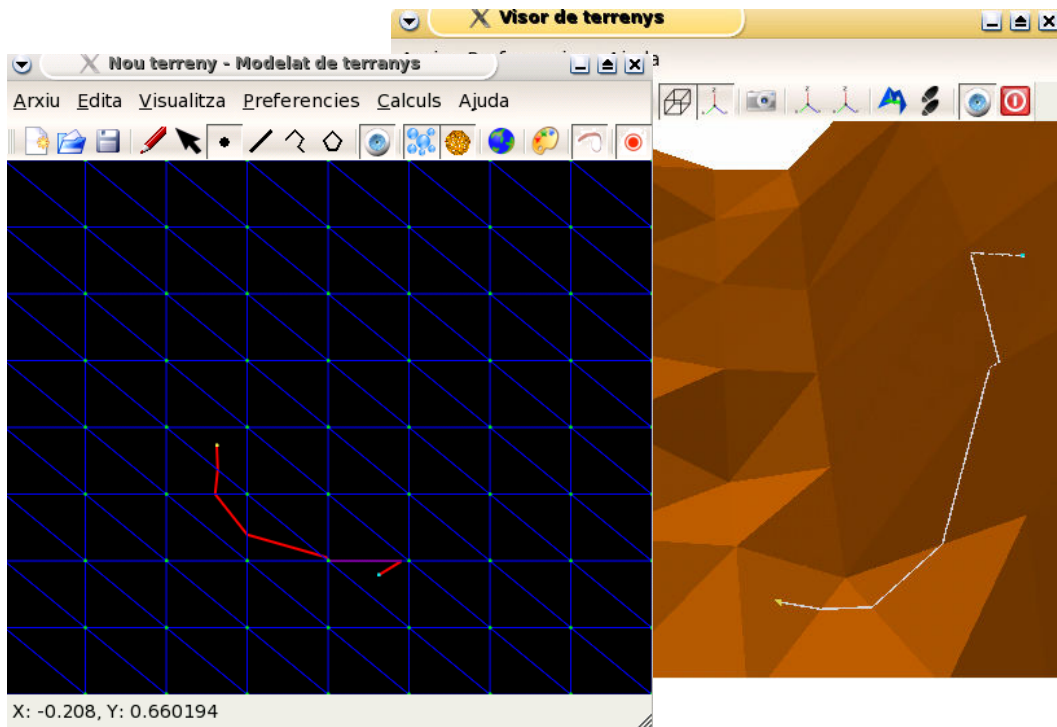
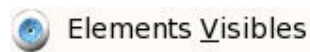


Figura 8.27 : Exemple de trobar camí més curt entre punt i seu

8.3.15 Visualitzar elements visibles

Un altre aspecte que l'aplicació ens permet realitzar és el fet de poder visualitzar en tot moment els elements de distàncies que anem inserint en el terreny, ja siguin seus, punts de camí mínim o elements restringits (vèrtexs de la triangulació). Així doncs, tindrem una opció, per defecte activa, per tal de poder visualitzar tots aquests elements "extres" que anem afegint al terreny per tal que ens ajudin a resoldre la nostra problemàtica. Per tal d'activar o desactivar aquesta opció ho farem de la següent forma:

- Des del menú principal accedint a la opció **Elements Visibles** del menú **Visualitza**, o bé prement el botó *Elements Visibles* de la barra d'eines de la finestra principal



Comentar que el *Visor 3D* també disposa d'una opció igual que aquesta, però que tal i com explicarem més endavant, és totalment independent de la de *l'Editor 2D*. És a dir, podem visualitzar els elements visibles en *l'Editor 2D* i no visualitzar-los en el *Visor 3D* i viceversa.

8.3.16 Visualitzar Diagrames de Voronoi

Una altra de les funcionalitats més utilitzades serà la de visualitzar el *Diagrama de Voronoi* per a un conjunt de seus. Aquesta funcionalitat consistirà, tenint un terreny ja propagat amb totes les finestres, en veure per cada punt del terreny quina és la seu que té més a prop, a partir d'un color, que definirà l'abast de punts mínims que té aquella seu d'aquell determinat color. És a dir, tots els punts que es trobin en la regió definida per un mateix color, tindran la seu continguda en aquella regió com la més propera. Per tal de poder realitzar aquesta opció, haurem de seguir els següents passos:

- Des del menú principal activar la modalitat de treball **Distàncies** del menú **Calculs**, o bé prémer la combinació de tecles **CTRL + D**.

Distàncies

Ctrl+D

- Des del menú principal accedir a la opció **Inserir punt (seu)** del menú **Edita**, o bé prémer el botó *Inserir Punt (seu)* de la barra d'eines de la finestra principal, o bé prémer la combinació de tecles **CTRL + P**. (realitzar aquesta opció diverses vegades per tal d'inserir més d'una seu)

Inserir Punt (seu)

Ctrl+P

- Des del menú principal accedir a la opció **Propagar Terreny** del menú **Calculs**, o bé prémer el botó *Propagar Terreny* de la barra d'eines de la finestra principal

Propagar terreny

- Des del menú principal accedir a la opció **Diagrames de Voronoi** del menú **Visualitza**, o bé prémer el botó *Diagrames de Voronoi* de la barra d'eines de la finestra principal, o bé prémer la combinació de tecles **CTRL + V**.



Diagrames de Voronoi

Ctrl+V

Una vegada haguem realitzat aquests passos, visualitzarem per pantalla el *Diagrama de Voronoi* per a aquell terreny amb aquell conjunt de seus, tal i com es pot observar en la *Figura 8.28*.

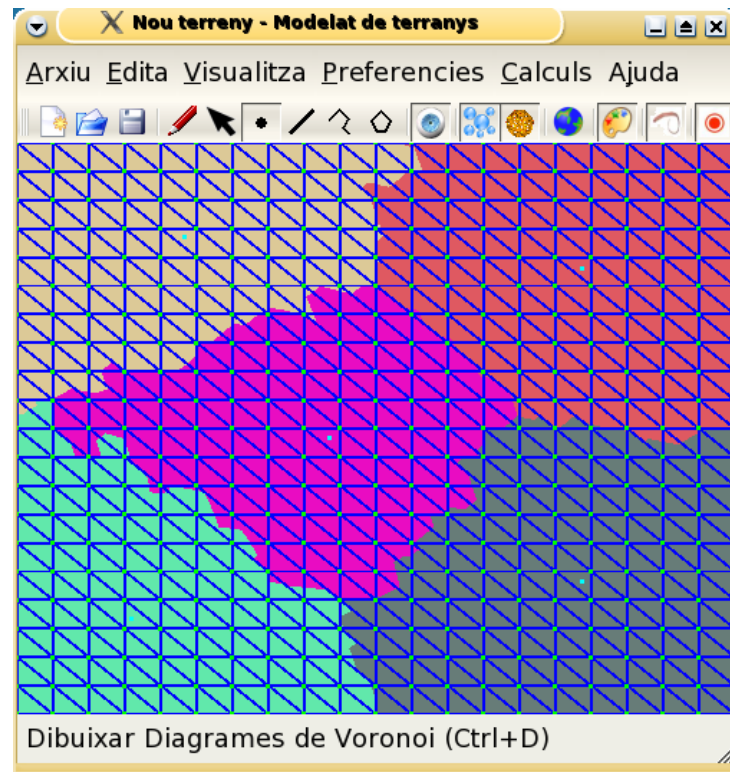


Figura 8.28 : Diagrama de Voronoi per a un conjunt de 5 seus

8.3.17 Ajuda

L'aplicació disposa també d'informació addicional sobre ella. Així doncs, tindrem el menú **Ajuda** accessible des del menú principal de l'*Editor 2D*. En aquest cas, tenim la possibilitat d'obtenir informació sobre:

- **Quant a...:** El contingut de la finestra actual, és a dir, de l'*Editor 2D*

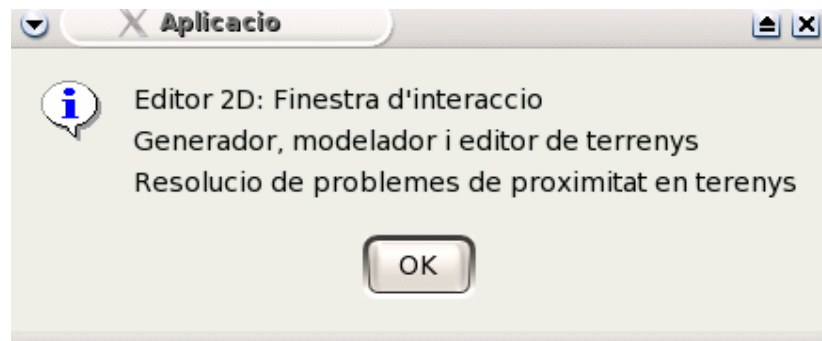


Figura 8.29 : Informació sobre l'Editor 2D

- **Quant a Qt:** Conté informació sobre les llibreries *Qt*, eina utilitzada en el disseny

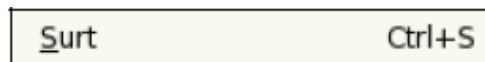


Figura 8.30 : Informació sobre Qt


8.3.18 Sortir de l'aplicació

Com amb qualsevol aplicació, tenim la possibilitat de tancar-la. Per poder sortir del generador i editor de terrenys, tenim diverses possibilitats:

- Des del menú principal accedir a la opció **Surt** del menú **Arxiu**

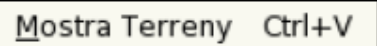


- Utilitzant la combinació de tecles **CTRL + S**, tal i com es mostra al costat de l'operació **Surt** del menú **Arxiu**

- A través de la  que es troba en la part superior dreta de l'Editor 2D

8.3.19 Mostrar / Amagar Visor 3D

Des de l'Editor 2D tenim la possibilitat de visualitzar el terreny en 2.5D. Aquesta operació consisteix en obrir la finestra del Visor 3D. Podrem fer-lo aparèixer a través de l'opció **Mostra Terreny** del menú **Visualitza**, o bé, utilitzant la combinació de tecles **CTRL + V**.



En activar l'opció ens apareixerà la finestra del Visor 3D, formada per un menú principal, una barra d'eines i l'espai de treball. La missió del visualitzador és permetre veure el model tridimensional del terreny visualitzat en l'Editor 2D des de diferents punts de vista.

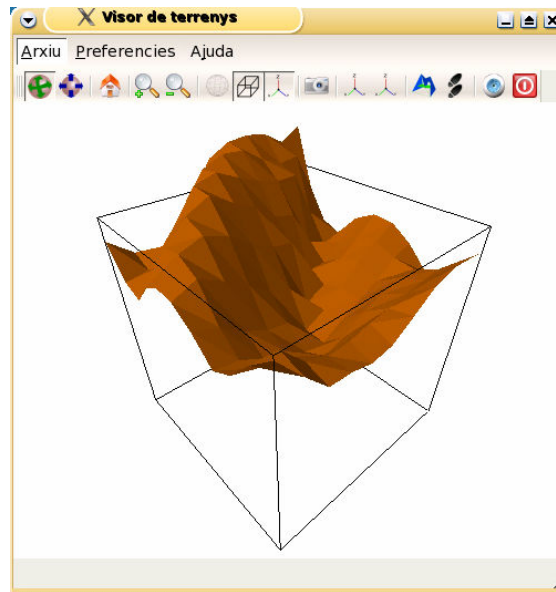



Figura 8.31 : Exemple de visualització sobre el Visor 3D

Tot seguit es descriuran les funcionalitats del *Visor 3D* que tan sols afectaran al terreny visualitzat en el visualitzador.

8.3.19.1 Opcions de navegació: moviment de la càmera

Des del **Visor 3D** podrem visualitzar el terreny des de diferents perspectives. Per fer-ho, disposem de dos tipus de moviments que permeten moure la càmera de posició, tot i que tan sols podrà estar activat un dels dos al mateix temps:

- **Rotació:** A partir del botó  *Rotar Camera* o amb la combinació de tecles **CTRL + R** podrem accionar el mode de rotació. A continuació podrem rotar la càmera de dues maneres diferents:
 - *Amb el moviment del ratolí:* En aquest cas, si premem el botó esquerre del ratolí i el mantenim pres, ens movem de dreta a esquerra o a l'invers per sobre de l'espai de treball del *Visor 3D*. D'aquesta manera aconseguirem que la càmera giri al voltant del terreny sobre l'eix Y. D'altra banda, si fem que el ratolí es mogui de dalt a baix o a l'invers, aconseguirem que la càmera giri al voltant del terreny sobre l'eix X
 - *A través dels controls de direcció del teclat:* Mitjançant les tecles que apareixen en la *Figura 8.32* podrem realitzar les rotacions equivalents a les realitzades amb el ratolí.

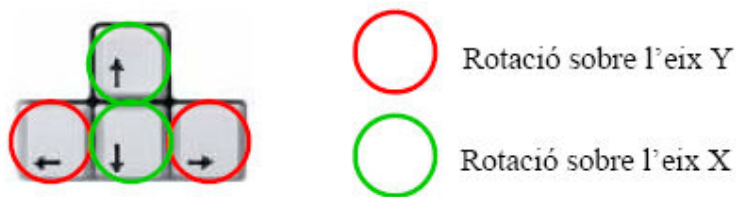




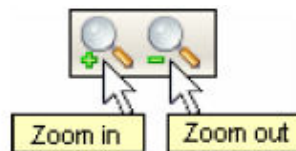
Figura 8.32 : Tecles de rotació

- **Translació:** A partir del botó  *Desplaçar càmera* o amb la combinació de tecles *CTRL + D* podrem accionar el mètode de translació. Aquesta operació permetrà desplaçar la càmera tant en horitzontal com en vertical. En aquest cas, l'acció tan sols es podrà dur a terme amb el moviment del ratolí. De la mateixa manera que abans, caldrà prémer el botó esquerre del ratolí i mantenint-lo premut, realitzar el desplaçament que es desitgi arrossegant el punter del ratolí fins on es vulgui.

Existeix també la possibilitat de tornar a la posició inicial després d'haver fet qualsevol moviment. A través del botó  *Iniciar vista* de la barra d'eines o de la tecla *Home*, simbolitzat amb una casa, podrem tornar a situar la càmera en el punt de vista per defecte.

8.3.19.2 Realitzar zoom



Des del **Visor 3D** podrem realitzar zoom sobre el terreny tant per allunyar la càmera (*Zoom -*) com per apropar-la (*Zoom +*) del model. Mitjançant els botons *Apropar* (o tecla *+*) i *Allunyar* (o tecla *-*) de la barra d'eines, podrem realitzar el control del zoom. Una altra possibilitat és fer-ho a través del ratolí. En el cas que disposem d'un ratolí amb una roda central, el zoom podrà ésser controlat a través del moviment d'aquesta.



A través del zoom serem capaços d'observar amb més detall el terreny d'aquelles parts que vulguem. La combinació entre els moviments de rotació i translació, amb el zoom, ens permet poder veure el terreny amb el detall desitjat des de les perspectives escollides.

8.3.19.3 Opcions de visualització

Disposem de diverses opcions de visualització de l'entorn del terreny. A través de diversos botons de la barra d'eines del **Visor 3D** podrem dur a terme aquests canvis de configuració. Totes elles consisteixen en l'activació o desactivació de la visió d'un cert element. Tenim les següents opcions:

- **Veure límits de l'espai:** Botó  que ens permet visualitzar o no el cub englobant, que indica les mides de configuració del terreny introduïdes en la finestra principal
- **Veure eixos de coordenades:** Botó  a partir del qual podrem activar o desactivar la visualització dels eixos de coordenades que ens permeten tenir un punt de referència sobre la situació del punt (0,0,0)

Per defecte, els dos botons es troben activats en iniciar l'aplicació.

8.3.19.4 Guardar informació en format gràfic

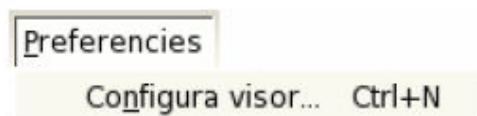
De la mateixa manera que amb la finestra principal de l'*Editor 2D*, també disposem de la possibilitat de capturar en format d'imatge el contingut de l'espai de treball del **Visor 3D**. Simplement caldrà accionar el botó *Captura Imatge*, o la combinació de tecles *CTRL + I* o la opció **Captura Imatge** del menú **Arxiu**, i ens apareixerà la mateixa finestra de diàleg que en el cas de la captura de l'*Editor 2D* (Figura 8.19)



A continuació, caldrà realitzar el mateix procés, és a dir, indicar el nom i format del fitxer que contindrà la imatge del terreny 3D. Aquesta imatge contindrà l'aspecte del *Visor 3D* tal i com estava en el moment de fer la captura.

8.3.19.5 Modificar configuració del Visor 3D

Alguns dels elements del visualitzador podran ésser configurats per l'usuari. D'aquesta manera podrà personalitzar part del **Visor 3D**. Per fer-ho, haurà d'accedir a la opció **Configuració del visor** situada en el menú **Preferències**.




Accionant aquesta opció, apareixerà la finestra de la *Figura 8.33*, a través de la qual podrem modificar el color de fons del visor, el color del model, el color del cub englobant i el color de cadascun dels eixos de coordenades.



Figura 8.33 : Finestra de configuració dels colors del Visor 3D

En qualsevol dels casos, la selecció del color es durà a terme a través d'un simple botó. Un cop escollida la configuració desitjada, tan sols caldrà prémer el botó *Accepta* perquè aquesta sigui aplicada. En el cas que vulguem anul·lar la operació, simplement accionarem el botó *Cancel·la* i tot continuarà igual, sense cap canvi.

8.3.19.6 Veure finestres

Una altra possibilitat que ens ofereix el **Visor 3D** és la possibilitat de poder visualitzar en el terreny les finestres que s'hi han creat una vegada hem realitzat la opció de *Propagar Terreny*. Per tal de veure les finestres del terreny, tant sols haurem d'activar el botó  *Finestres* i immediatament visualitzarem per la finestra del *Visor 3D* el terreny amb totes les finestres de les quals disposa, tal i com es pot observar a la *Figura 8.34*.

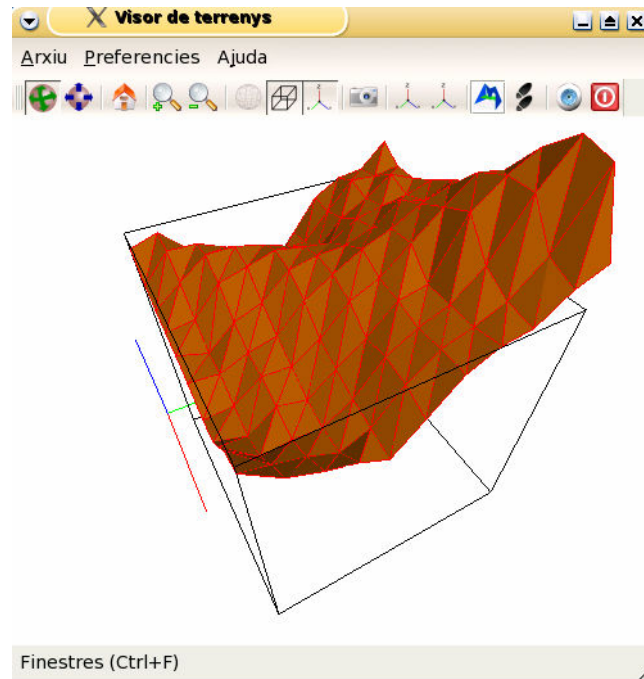


Figura 8.34 : Visualització al Visor 3D de les finestres que hi ha al llarg del terreny

8.3.19.7 Visualitzar elements visibles

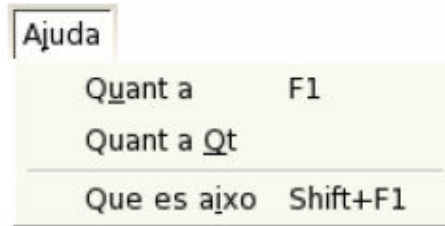
Un altre aspecte que l'aplicació ens permet realitzar és el fet de poder visualitzar en tot moment els elements de distàncies que anem inserint en el terreny, ja siguin seus, punts de camí mínim o elements restringits (vèrtexs de la triangulació). Així doncs, tindrem una opció per tal de poder visualitzar tots aquests elements "extres" que anem afegint al terreny per tal que ens ajudin a resoldre la nostra problemàtica. Amb aquesta opció, també hem de poder visualitzar el camí més curt entre una seu i un punt. Per tal d'activar o desactivar aquesta opció ho farem de la següent forma:

- Prement el botó  *Elements Visibles* de la barra d'eines de la finestra principal

Comentar que *l'Editor 2D* també disposa d'una opció igual que aquesta, però que és totalment independent de la del *Visor 3D*. És a dir, podem visualitzar els elements visibles en *l'Editor 2D* i no visualitzar-los en el *Visor 3D* i viceversa.

8.3.19.8 Ajuda

Existeix també informació d'ajuda accessible des del menú **Ajuda** del menú principal



Tenim la possibilitat d'obtenir informació sobre:

- **Quant a...:** El contingut de la finestra actual, és a dir, del *Visor 3D*

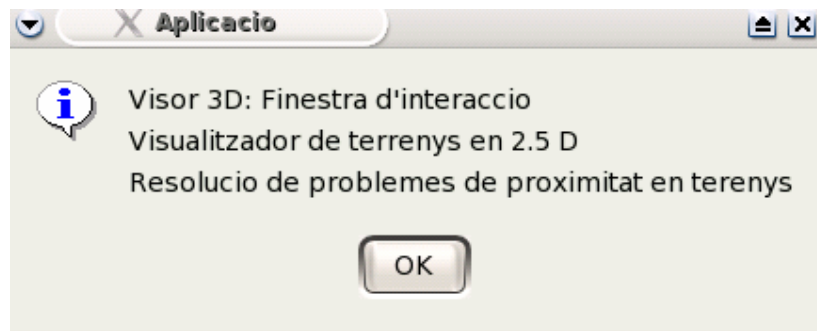


Figura 8.35 : Informació sobre el Visor 3D

- **Quant a Qt:** Conté informació sobre les llibreries *Qt*, eina utilitzada en el disseny




Figura 8.36 : Informació sobre Qt

8.3.19.9 Tancar el Visor 3D

La finestra corresponent al *Visor 3D* podrà ésser tancada. Aquesta operació no significarà el tancament del programari ja que el *Visor 3D* és una funcionalitat més de l'aplicació. Podrem amagar o tancar el visualitzador de diferents maneres:

- Selecciónant el botó **Sortir** de la barra d'eines del *Visor 3D*



- Com amb qualsevol finestra, a través de la  que es troba a la part superior dreta del visualitzador

Conclusions

En el *Capítol 1* d'aquest document presentàvem una descripció detallada de tots els objectius que havíem d'assolir per tal de poder implementar els algorismes de resolució de problemes de proximitat en terrenys, havent-los d'integrar a una interfície per tal que ens permetés realitzar-ne tant el càlcul com la visualització tant en 2D com en 3D.

Per tal d'assolir aquests objectius hem hagut de resoldre tota una sèrie de problemes que ens han anat sorgint al llarg del projecte, doncs recordem que inicialment varem haver de passar per un procés d'aprenentatge de la interfície i de les llibreries, sense el qual no haguéssim pogut portar a terme aquest projecte. Afortunadament, els objectius sembla que s'han realitzat tot i que som conscients que hi ha millores per dur a terme i per ampliar (comentades en part en el *Capítol 7*) i amb les quals es seguirà treballant en un futur per tal de millorar les prestacions del programa.

En aquest apartat final, presentarem les conclusions extretes del desenvolupament d'aquest projecte, fent referència tant a les conclusions respecte el compliment dels objectius (funcionalitats) que ens havíem marcat com als objectius d'aprenentatge. També veurem si els terminis s'han complert i quines han estat les fases de desenvolupament dutes a terme a l'hora de realitzar aquest projecte.

L'objectiu principal del projecte era desenvolupar una aplicació per a la **resolució de diversos problemes de proximitat en terrenys**. Aquest objectiu, tal i com s'ha pogut comprovar al llarg de tota la memòria, ha estat complet de forma satisfactòria. A més a més, tot el conjunt de requeriments definits des d'un principi han estat assolits en l'aplicació final. Si recordem els objectius bàsics que ens havíem marcat per tal d'arribar a l'objectiu final, tal i com havíem establert en el *Capítol 1*, veurem com aquests també s'han dut a terme de forma satisfactòria. Aquests eren:

- Construir una interfície que ens permeti escollir seus (punts) sobre un terreny.
- Dissenyar i implementar diferents algorismes per tal de determinar camins òptims sobre terrenys.
- Calcular la distància geodèsica entre un punt del terreny i la seva seu més propera.
- Poder visualitzar tant en 2D com en 3D camins òptims sobre terrenys.
- Calcular i visualitzar Diagrames de Voronoi sobre terrenys.

El desenvolupament d'aquest projecte, però, ha permès aconseguir dos tipus d'objectius. Per una banda els relacionats amb els requeriments de l'aplicació i per l'altre els referents a l'aprenentatge tant de temes que tenen a veure amb la geometria, com el diferent software o llibreries utilitzades durant al llarg del projecte. Pel que fa als **requeriments de l'aplicació** hem complert tots els objectius plantejats des de bon principi. Hem aconseguit, entre d'altres:

- Disposar d'una interfície gràfica d'usuari amigable que ens permeti inserir seus (punts) en un terreny, així com també ens permet afegir elements restringits (punts, vèrtexs, polígons, poligonals) .
- Dissenyar i implementar algorismes per tal de determinar el camí més curt sobre terrenys entre un o diversos punts i la seva seu més propera, a partir de la creació i propagació de finestres.
- Calcular la distància geodèsica entre un punt i la seva seu més propera
- Poder visualitzar la reconstrucció del camí òptim entre el punt i la seu tant en 2D com en 3D de forma instantània
- Calcular i visualitzar *Diagrames de Voronoi* sobre terrenys per a un conjunt de seus, sabent cada punt del terreny quina és la seu que té més propera
- Poder visualitzar, generar, editar i emmagatzemar terrenys en diferents formats

Pel que fa a l'**aprenentatge**, també s'ha aconseguit satisfer els objectius inicials que ens havíem marcat. S'ha aconseguit ampliar els coneixements relacionats amb els conceptes següents:

- Definicions i conceptes de Geometria Computacional
- Algorismes geomètrics relacionats amb els problemes de proximitat en terrenys (Algorisme de Dijkstra, Diagrames de Voronoi, ...)
- Algorismes bàsics per el treball amb figures geomètriques
- Estructures geomètriques de dades (DCEL)
- Llibreries de domini públic (*OpenGL*, *Qt*, *CGAL*, ...)
- Eines de software pel desenvolupament del projecte (*KDevelop*, *QtDesigner*, ...)
- Adquisició d'experiència en desenvolupar projectes i en comunicació amb els altres membres del grup de recerca

Així doncs, malgrat els diversos problemes que ens han anat sorgint durant el desenvolupament del projecte, podem dir que s'han aconseguit assolir els objectius inicials, tant de requeriments com d'aprenentatge. Malgrat tots aquests objectius que hem complert satisfactòriament, som conscients que hi ha millores a realitzar en un futur per tal de millorar les prestacions actuals de la interfície.

Finalment, comentar que el desenvolupament d'aquest projecte ha passat per diferents etapes, intentant seguir una *metodologia orientada a objectes basada en UML*. Com que la metodologia és iterativa i incremental, hem pogut anar evolucionant i millorant les etapes anteriors a través de les diverses fases. Les diferents **fases** que hem seguit, de forma més o menys exacta, i que ens han permès poder complir amb els terminis previstos són les següents:

- *Aprenentatge de la interfície inicial*
- *Aprenentatge de les llibreries utilitzades*
- *Estudi dels conceptes previs relacionats amb els problemes de proximitat en terrenys*
- *Descripció dels requeriments del sistema*
- *Anàlisi del sistema*
- *Disseny del sistema*

- *Implementació i Proves*
- *Documentació*

Agraïments

Arribats a aquest punt de la memòria, m'agradaria dedicar unes línies a aquelles persones que m'han ajudat i m'han donat suport durant el transcurs d'aquest projecte i sense les quals no hagués estat possible poder-lo acabar amb èxit.

Primer de tot vull agrair al professor Joan Antoni Sellarès que confiés amb mi a l'hora de realitzar aquest projecte i que m'assignés una plaça per realitzar una tasca dins el seu grup de recerca d'investigació, i que m'envoltés d'un grup de gent qualificada que coneixien el tema, fet que ha permès que em sentís recolzat en tot moment.

I d'altra banda vull donar les gràcies també a la Marta Fort, que m'ha ajudat en tot moment, sempre ha estat al meu costat i m'ha donat sempre suport i idees per poder tirar endavant aquest projecte, i sense la qual no hagués pogut acabar amb èxit. Des d'aquí li agraeixo tot el que ha fet per mi i li desitjo molta sort de cares al seu doctorat, que espero que pugui acabar tant aviat com sigui possible.

Bibliografia

- [CH96] J. Chen, Y. Han, Shortest paths on a polyhedron, *Internat. J. Comput. Geom. Appl.* (1996), Vol. 6., pp. 127--144.
- [KO00] B. Kaneva, J. O'Rourke, An Implementation of Chen and Han's Shortest Paths Algorithm, in *Proc. of the 12th Canadian Conference on Computational Geometry*, (2000), pp. 139--146.
- [Kapoor93] S. Kapoor, Efficient Computation of Geodesic Shortest Paths, *Proc. 32nd Annu. ACM Sympos. Theory Comput.*, (1999) pp. 770--779.
- [MMP87] J. Mitchell, D. Mount, H. Paparimitri, The discrete geodesic problem, *SIAM J. Computation* 6 (1987), pp. 647--668.
- [Mit00] J. S. B. Mitchell, Geometric shortest paths and network optimization. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of computational geometry*, Elsevier, (2000), pp. 633--701.
- [SSKGH05] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, H. Hoppe, Fast Exact and Approximate Geodesics on Meshes, In *ACM Transactions on Graphics (TOG)*, Proceedings of ACM SIGGRAPH, 24:3, pp. 553 - 560, 2005.
- [BKOO97] M. de Berg, M. van Kreveld, R. van Oostrum and M. Overmars. *Computational Geometry: algorithms and applications*. Springer, 1997
- [Shew97] J.R. Shewchuck *Delaunay Refinement Mesh Generation*. Phd thesis School of Computer Science – Carnegie Mellon University, 1997
- [PrepSha85] F. Preparata and M. Shamos. *Computational Geometry. An Introduction*. Springer, New York, 1985
- [Sell2003] Toni Sellarès. *Enginyeria del Programari Orientada a Objectes: Introducció a UML*, 2003

Adreces d'Internet

Sobre lliberies OpenGL:

<http://www.opengl.org>

Sobre lliberies Qt:

<http://www.trolltech.com>

Sobre lliberies CGAL:

<http://www.cgal.org>

Sobre software utilitzat:

<http://www.objectteering.com/>

<http://www.trolltech.com/products/qt/designer.html>

<http://www.kdevelop.org>

Sobre problemes resolt:

<http://portal.acm.org/citation.cfm?id=1073228>

<http://portal.acm.org/citation.cfm?id=1073204.1073228>

<http://www.it.uc3m.es/~prometeo/rsc/apuntes/encamina/encamina.html>

Índex de Figures

Figura 1.1 : Navegador de cotxe	11
Figura 1.2 : Combinació de dades per capes	11
Figura 1.3 : Exemples d'aplicacions	12
Figura 2.1 : Exemples utilitzats per calcular temps d'execució en la referència [SSKGH05]	15
Figura 2.2 : Esquema d'un terreny	16
Figura 2.3 : En els casos (a) i (b) els triangles interseccionen, en (c) existeix un polígon simple que no és un triangle i (d) és una triangulació correcte	17
Figura 2.4 : Definició d'aresta Delaunay	17
Figura 2.5 : Cadascuna de les arestes que formen l'envolupant convexa és Delaunay, ja que sempre és possible trobar un cercle buit que passi a través dels seus extrems	17
Figura 2.6 : Com que els punts no es troben en posició general ja que existeix una circumferència que passa per més de tres punts (b) existeixen diverses triangulacions Delaunay possibles, tal i com es mostra en la figura (a). En la figura (c) s'escull un subconjunt d'arestes Delaunay per formar una triangulació	18
Figura 2.7 : Si el triangle t no és Delaunay (a) almenys un dels vèrtexs de V està contingut pel circumcentre de t . En la figura (b) t és Delaunay	18
Figura 2.8 : A partir d'un núvol de punts (a) obtenim una triangulació de Delaunay (b) on tot triangle té un circumcentre buit	18
Figura 2.9 : Triangulació de Delaunay d'un núvol de punts amb els circumcentres buits i en vermell els circumcentres de cadascun d'ells que connectats formaran el Diagrama de Voronoi (b) també en vermell	19
Figura 2.10 : Esquema d'una subdivisió planar on e i $\text{Tw}(e)$ són les arestes orientades oposades d'una mateixa aresta	20
Figura 2.11 : Informació emmagatzemada al DCEL per cada aresta orientada	20
Figura 2.12 : Exemple d'estructura DCEL	21
Figura 2.13 : Exemple de l'algorisme de Dijkstra	23
Figura 2.14 : Exemple de Diagrama de Voronoi	24
Figura 2.15 : Exemple d'un vèrtex seu	25
Figura 2.16 : (a) Rajos emergents de la seu s a través de la finestra w al llarg de la tira de triangles oberts. (b) La posició de s a partir dels paràmetres de la finestra. (c) Finestra emergent d'un pseudosource s a distància σ de la seu original	27

Figura 2.17 : (a,b) Exemples de propagació de finestres que ens formen : (a) una nova finestra a un costat ; (b) dues noves finestres en dos costats diferents. (c) Es forma una nova finestra i un nou vèrtex pseudosource v	28
Figura 2.18 : (a) Dues finestres w_0 i w_1 que es solapen (amb els seus respectius pseudosources s_0 i s_1) i la seva intersecció $[b_0, b_1]$. (b) La formació de dues finestres disjunts a partir de dues finestres que es solapaven, amb els nous intervals $b_0' - b_1'$ per totes dues i amb les noves distàncies d	28
Figura 2.19 : Exemple de la interfície del KDevelop	37
Figura 2.20 : Exemple de la interfície del Objectteering UML Modeler	40
Figura 3.1 : Actors que interactuen amb el sistema	46
Figura 3.2 : Diagrama de cas d'ús de context del Sistema	47
Figura 3.3 : Refinament del cas d'ús Camí òptim entre seu i punt, on tindrem les opcions de calcular el camí òptim i visualitzar aquest camí. Tal i com veurem en capítols posteriors, aquest requeriment englobarà tota la part del càlcul de girs de desplegament, creació, propagació i intersecció de finestres, etc.....	48
Figura 3.4 : Refinament del cas d'ús Diagrames de Voronoi sobre Terrenys, on tindrem les opcions de Calcular el Diagrama de Voronoi i visualitzar aquest Diagrama de Voronoi	48
Figura 3.5 : Refinament del cas d'ús Editar Terreny, on durem a terme opcions diferents depenent de si estem tractant amb Elements de Distàncies (seus) o elements Restringits. La nostra part fa referència principalment a la inserció i eliminació d'elements de distàncies (seus)	49
Figura 3.6 : Refinament del cas d'ús Visualitzar Terreny, on disposarem de diferents opcions de visualització, com poden ésser la rotació de la càmera, la modificació del zoom i la modificació d'altres opcions de visualització del Terreny	49
Figura 3.7 : Diagrama d'activitat del cas d'ús Calcular camí òptim entre seu i punt ...	54
Figura 3.8 : Diagrama d'activitat del cas d'ús Visualitzar Diagrama de Voronoi sobre terreny	55
Figura 4.1 : Diagrama de classes	61
Figura 4.2 : Diagrama de seqüència Inserir seu al Terreny	69
Figura 4.3 : Diagrama de seqüència Camí òptim entre seu i punt	71
Figura 4.4 : Diagrama de seqüència Diagrama de Voronoi sobre terreny	73
Figura 5.1 : Interfície final de l'Editor 2D	76
Figura 5.2 : Interfície final del Visor 3D	79
Figura 6.1 : Terreny amb una seu inserida	83
Figura 6.2 : Finestres creades a partir del pas d'inicialització	85
Figura 6.3 : Propagació de finestres i pseudosources al llarg del terreny	86
Figura 6.4 : Punt de camí mínim (color groc) inserit en el terreny	87
Figura 6.5 : Exemple camí més curt entre seu i punt	88
Figura 6.6 : Exemple camí més curt entre seu i punt	88
Figura 6.7 : Exemple camí més curt entre seu i punt	89
Figura 6.8 : Terreny amb més d'una seu	90
Figura 6.9 : Exemple propagació de finestres per a un conjunt de seus	93

Figura 6.10 : Diagrama de Voronoi per al conjunt de seus de l'exemple anterior	94
Figura 6.11 : Diagrama de Voronoi per a un conjunt de 5 seus	95
Figura 6.12 : Diagrama de Voronoi per a un conjunt de 3 seus	95
Figura 6.13 : Joc de proves de la funció Trobar Source	97
Figura 6.14 : Joc de proves de la funció Inicialització	98
Figura 6.15 : Joc de proves de la funció Propagar PseudoSource	98
Figura 6.16 : Joc de proves de la funció Propagar Finestra	99
Figura 6.17 : Joc de proves de la funció Interseca	100
Figura 6.18 : Joc de proves de la funció No_Elimina	101
Figura 6.19 : Joc de proves de la funció Actualitzar Llista Finestres	101
Figura 8.1 : Finestra principal o Editor 2D	107
Figura 8.2 : Menú principal de l'Editor 2D	107
Figura 8.3 : Menú Arxiu de l'Editor 2D	108
Figura 8.4 : Menú Edita de l'Editor 2D	108
Figura 8.5 : Menú Visualitza de l'Editor 2D	109
Figura 8.6 : Menú Preferències de l'Editor 2D	109
Figura 8.7 : Menú Càlculs de l'Editor 2D	109
Figura 8.8 : Menú Ajuda de l'Editor 2D	109
Figura 8.9 : Barra d'eines de l'Editor 2D	110
Figura 8.10 : Visualitzador o Visor 3D	111
Figura 8.11 : Menú principal del Visor 3D	111
Figura 8.12 : Menú Arxiu del Visor 3D	112
Figura 8.13 : Menú Preferències del Visor 3D	112
Figura 8.14 : Menú Ajuda del Visor 3D	112
Figura 8.15 : Barra d'eines del Visor 3D	112
Figura 8.16 : Terreny generat de forma aleatòria	117
Figura 8.17 : Finestra de diàleg per tal de carregar un terreny amb algun dels formats suportats per l'aplicació	118
Figura 8.18 : Finestra de diàleg per tal de desar un terreny	119
Figura 8.19 : Finestra de diàleg per tal de guardar com a imatge el terreny	120
Figura 8.20 : Missatge d'error al intentar inserir un segment en el terreny que actui com a seu	121
Figura 8.21 : Finestra de diàleg per tal d'introduir l'alçada del punt	122
Figura 8.22 : Exemple d'inserció punt camí mínim (color groc) en el terreny	125
Figura 8.23 : Finestra de diàleg que mostra la informació (coordenades X, Y i Z) del punt seleccionat	125
Figura 8.24 : Finestra de configuració de les dimensions de l'espai a l'Editor 2D.....	127
Figura 8.25 : Finestra de configuració dels colors a l'Editor 2D	128

Figura 8.26 : Exemple de propagació de finestres per a un conjunt de seus	130
Figura 8.27 : Exemple de trobar camí més curt entre punt i seu	131
Figura 8.28 : Diagrama de Voronoi per a un conjunt de 5 seus.....	133
Figura 8.29 : Informació sobre l'Editor 2D	133
Figura 8.30 : Informació sobre Qt	134
Figura 8.31 : Exemple de visualització sobre el Visor 3D	135
Figura 8.32 : Tecles de rotació.....	136
Figura 8.33 : Finestra de configuració dels colors del Visor 3D	138
Figura 8.34 : Visualització al Visor 3D de les finestres que hi ha al llarg del terreny	139
Figura 8.35 : Informació sobre el Visor 3D	140
Figura 8.36 : Informació sobre Qt	140
Figura B.1 : Representació d'un actor	160
Figura B.2 : Representació d'un cas d'ús	160
Figura B.3 : Comunicació entre actor i cas d'ús	161
Figura B.4 : Relació d'herència entre actors	161
Figura B.5 : Exemple d'un diagrama de casos d'ús	162
Figura B.6 : Representació d'inici d'un diagrama d'activitat	163
Figura B.7 : Representació de fi d'un diagrama d'activitat	163
Figura B.8 : Representació d'activitat d'un diagrama d'activitat	163
Figura B.9 : Representació de branques d'un diagrama d'activitat	163
Figura B.10 : (a) Forquilla d'inici o inici d'activitats en paral·lel. (b) Forquilla de fi o unió de retrobament de les activitats	164
Figura B.11 : Exemple de diagrama d'activitat	164
Figura B.12 : Representació d'una classe	165
Figura B.13 : Representació d'una classe abstracta	166
Figura B.14 : Representació d'un paquet	166
Figura B.15 : Relació Generalització/Especialització	167
Figura B.16 : Associació per composició (Cinema – Pel·lícula) i per agregació (Cinema – Guixeta)	168
Figura B.17 : Relació de dependència	168
Figura B.18 : Representació d'una nota.....	169
Figura B.19 : Representació d'un objecte d'un diagrama de seqüència	169
Figura B.20 : Representació d'una classe entitat	170
Figura B.21 : Representació d'una classe de control	170
Figura B.22 : Representació d'una classe de frontera	170
Figura B.23 : Exemple de diagrama de seqüència	171

Annex A

Metodologia basada en UML

Una metodologia defineix qui fa què, quan i com fer-ho per arribar a un cert objectiu. En el cas d'aquest projecte, l'objectiu serà arribar a desenvolupar un software requerit.

En aquest projecte s'ha utilitzat una *metodologia Orientada a Objectes (OO)* basada en **UML** pel desenvolupament del software.

Una metodologia de desenvolupament de software OO consta de:

- Conceptes i diagrames
- Etapes i definició de lliuraments en cadascuna d'elles
- Activitats i recomanacions

La **metodologia basada en UML** presentada en aquest annex és un extracte de metodologies existents. En particular les tres metodologies pioneres en OO són les següents: *Object Oriented Design*, *Objectory* i *Object Modeling Technique*. Aquestes metodologies van ésser fusionades a finals del 1997 en una única, basada en la notació UML, anomenada *UP (Unified Process)*.

Tot seguit es presenta un resum de les principals etapes i activitats d'aquesta metodologia.

A.1 Etapes i activitats en el desenvolupament OO sobre UML

La metodologia proposa les següents etapes a tenir en compte en el desenvolupament de software:

- Descripció de requeriments i anàlisi del sistema
- Disseny del sistema
- Disseny detallat
- Implementació i proves

En cadascuna d'aquestes etapes és ressalten les activitats més importants i els documents que s'esperen al final de cadascuna d'elles.

A.1.1 Descripció de requeriments i anàlisi del sistema

En aquesta etapa cal definir clarament tot allò que desitja l'usuari i la forma en la qual se li presentarà la solució que busca. S'iterarà sobre aquesta etapa si al validar és decideix que cal més informació.

Les activitats tècniques d'aquesta etapa són:

1. Identificar els Casos d'ús del sistema

- a. Determinar els actors del sistema
- b. Determinar els casos d'ús per cadascun dels actors definits
- c. Trobar les relacions entre els actors i els casos d'ús

2. Detallar els Casos d'ús

- a. Descriure entrada i sortida del cas d'ús
- b. Fer una descripció detallada del cas d'ús (Descripció textual del seu objectiu, errors i possibles excepcions)
- c. Relacionar el cas d'ús amb la interfície que el representarà
- d. Especificar el diàleg que dona solució al cas d'ús

3. Definir una interfície inicial del sistema

- a. Dibuixar les pantalles d'interacció per cadascun del actors/usuaris
- b. Especificar el diàleg que dona solució al cas d'ús. Aquest diàleg és pot especificar de diverses maneres depenent de la complexitat de la interfície definida (en aquesta etapa és suggereix escollir el mínim nivell de detall possible per tal de donar més llibertat de disseny en les etapes posteriors):

- i. Mitjançant una descripció textual del seu funcionament
- ii. Mitjançant un diagrama d'interacció que mostra la seqüència d'operacions entre els objectes de la interfície i els actors involucrats
- iii. Mitjançant diagrames d'estat on és mostren clarament els estats de la interfície
- iv. Mitjançant un prototip funcional, en termes de la interacció amb l'usuari

4. *Desenvolupar el model del domini*

Aquesta informació és representada en un diagrama d'estructura estàtica de classes. El seu procés és el següent:

- a. Identificar classes
- b. Identificar atributs
- c. Identificar associacions
- d. Identificar missatges
- e. Identificar relacions de herència
- f. Identificar restriccions del model
 - i. Valors possibles i no possibles dels atributs
 - ii. Valors permesos per a les associacions
 - iii. Restriccions que relacionen dos o més atributs o relacions
- g. Identificar paquets

5. *Validar els models*

- a. Validar restriccions
- b. Validar atributs i missatges
- c. Desenvolupar diagrames d'interaccions per a la variant per defecte de cada cas d'ús, utilitzant els objectes del model del domini i els seus missatges.
- d. Validar els diagrames d'interacció
- e. Validar amb l'expert del domini
 - i. L'estructura del domini
 - ii. La funcionalitat esperada del sistema
 - iii. Els diagrames d'interacció descrits com a detall dels cas d'ús
- f. Validar amb un usuari representatiu de cadascun dels actors
 - i. La funcionalitat esperada per a l'actor en particular
 - ii. Els diagrames d'interacció descrits com a detall dels cas d'ús
 - iii. La interfície dissenyada i el diàleg descrit

Els següents són els documents a lliurar:

1. *Casos d'ús inicials:*

- Requeriments més importants del sistema
- Usuaris i sistemes externs en comunicació

- Especificació de requeriments
- 2. *Esborrany d'interfície*: Presentacions inicials pels diferents usuaris, de la forma de com solucionar els seus requeriments.
- 3. *Model del domini de l'aplicació*: Classes, relacions entre classes i especificacions.

A.1.2 Disseny del Sistema

En aquesta etapa, si el sistema és suficientment gran, es defineix una subdivisió en aplicacions del sistema i la forma de comunicació amb els sistemes ja existents amb els quals ha de interactuar.

Tenim aquí una única activitat tècnica que es *Identificar l'arquitectura del sistema* i els documents a lliurar són la versió inicial dels *Diagrames d'execució*.

Per tal de complementar la tasca *d'Identificar l'arquitectura del sistema* es segueixen els següent passos:

1. Definir els components del sistema, les aplicacions i la seva ubicació. Representar-los per mig de nodes, components i objectes actius dintre dels nodes.
2. Definir mecanismes de comunicació. Expressar-los mitjançant associacions de dependència entre els nodes, components o aplicacions i, si es conegut, afegir-hi un estereotip per a definir el protocol de comunicació requerit. Afegir-hi notes amb restriccions, rendiment esperat i més a més detalls de les connexions.
3. Particularitzar els casos d'ús a l'arquitectura plantejada. Refinar els casos d'ús ja existents de l'etapa anterior per tal d'adequar-los a l'arquitectura plantejada.
4. Validar l'arquitectura. Comprovar la validesa tècnica, econòmica i organitzacional de la proposta.

Els Diagrames d'execució que documenten aquesta etapa inclouen: Processadors, processos, mecanismes de comunicació i descripció detallada.

A.1.3 Disseny detallat

En aquesta etapa s'adequa l'anàlisi de les característiques específiques de l'ambient d'implementació i es completen les diferents aplicacions del sistema amb els models de control, interfície o comunicacions, segon sigui el cas.

Les activitats tècniques d'aquesta etapa són:

- 1. Afegir detalls d'implementació al model del domini**
 - a. Completar el detall de les classes:
 - i. Tipus dels atributs
 - ii. Atributs i mètodes de classe
 - iii. Disseny d'associacions
 - iv. Completar els mètodes
 - b. Incorporar patrons de disseny
 - c. Subdividir en paquets
 - d. Definir excepcions
 - e. Completar el comportament de les classes: Constructors, destructors, modificadors, consultors.
 - f. Adequar el model a les característiques del llenguatge de programació
 - g. Avaluar eficiència
 - h. Validar el sistema
- 2. Desenvolupar el model d'interfície:** Enllaçar les classes d'interfície amb les classes del model del domini

- 3. Desenvolupar els models de control, persistència i comunicacions**

Els documents lliurats són els següents:

- 1.** Diagrames de classes i paquets, amb el detall de la implementació
- 2.** Diagrames d'interacció amb el detall de les operacions més importants del sistema
- 3.** Diagrames d'estat i/o activitats par a les classes concurrents o complexes

A.1.4 Implementació i proves

És en aquesta etapa on es desenvolupa el codi. Les activitats tècniques especificades per aquesta etapa són les següents:

- 1. Definir estàndards de programació**
 - a. Conèixer i adequar estàndards de programació al llenguatge
 - b. Definir estructura de directoris
 - c. Dissenyar makefiles
- 2. Codificació i proves unitàries**
- 3. Proves de mòduls i del sistema**

Els documents lliurats són:

1. Codi font
2. Suport de proves unitàries
3. Documentació del codi

A.2 Procés de desenvolupament de software OO

Informalment, un procés de desenvolupament de software descriu una manera per a tractar la construcció, desenvolupament i manteniment del software. Els processos orientats a objectes són **iteratius** i **incrementals**, per tant la definició de requeriments, l'anàlisi i el disseny no queden completats fins que comença la fase de implementació. Cadascuna de les iteracions porta a l'elecció d'un petit subconjunt de requeriments i, ràpidament, dissenyar, implementar i provar. En les primeres iteracions, l'elecció dels requeriments i el disseny podrien no ser exactament el que es desitja. Però el fet de poder realitzar una petita iteració amb rapidesa, abans de capturar tots els requeriments i que el disseny complet s'hagi definit d'una manera especulativa, ens porta a una ràpida retroalimentació dels usuaris, desenvolupadors i proves. Per consegüent, la feina es desenvolupa a través d'una sèrie de cicles estructurats de *construir-retroalimentar-adaptar*.

Els beneficis d'un desenvolupament iteratiu inclouen:

- Mitigació tan aviat com es pugui dels alts riscos (tècnics, requeriments, objectius, etc.).
- Visible progrés en les primeres etapes.
- La retroalimentació es farà el més aviat possible, compromís dels usuaris i adaptació, que ens porta a un sistema refinat que s'ajusta més a les necessitats reals del personal involucrat.
- Gestió de la complexitat; l'equip no es veu aclaparat per la "paràlisi de l'anàlisi" o passos molt llargs i complexos.
- El coneixement adquirit en una iteració es pot utilitzar metòdicament per millorar el propi procés de desenvolupament, iteració a iteració.

Annex B

Nomenclatura UML

Durant el procés d'anàlisi s'han realitzat una sèrie de diagrames que segueixen la mateixa nomenclatura UML. El llenguatge de modelat unificat (UML) és un llenguatge de propòsit general per a la construcció de software orientat a objectes que ha estat proposat com a estàndard per l'Object Management Group (OMG). L'OMG, creat el 1989, és una organització no lucrativa dins la qual participen grans empreses de software, hardware, usuaris i consultors que té per objectiu l'elaboració d'estàndards relacionats amb el paradigma orientat a objectes. UML permet descriure un model d'anàlisi i disseny d'un sistema mitjançant diagrames construïts amb símbols que tenen:

- Regles semàntiques: Ens diuen que significa cada símbol i com interpretar-lo, tant quan es troba aïllat com quan es combina amb altres símbols en un diagrama.
- Regles sintàctiques: Ens diu com mostrar i combinar els símbols per a obtenir els diagrames del model.
- Regles pràctiques: Defineixen com utilitzar els símbols per a obtenir els diagrames del model de manera que siguin comprensible per altres persones.

Aquest llenguatge unificat de modelat ens permet visualitzar, especificar, construir i documentar els elements d'un sistema de programari des d'una perspectiva orientada a objectes. A continuació es detalla la notació utilitzada per tal de poder modelar i representar cadascun dels elements dels diagrames creats.

B.1 Nomenclatura utilitzada en els diagrames de casos d'ús

- **Actors:** Un actor és una entitat externa (persona, dispositiu, procés, subsistema, ...) que interactua amb el sistema interpretant un determinat rol. Aquests no són part del sistema que es construeix. La seva funció consisteix en fer entrar i rebre la informació del sistema. Els actors es representen per figures que esquemàticament representen una persona, com el següent:



Figura B.1 : Representació d'un actor

- **Casos d'ús:** Un cas d'ús descriu el comportament (funcionalitat) d'un sistema quan interactua amb un usuari extern (actor). Els casos d'ús disposen de les característiques següents:
 - Representen els requeriments funcionals del sistema.
 - Descriuen què fa un sistema, no com ho fa, des del punt de vista dels actors.
 - Dirigeixen tot el procés de desenvolupament d'un sistema.

Representarem els casos d'ús dins dels diagrames a través d'un oval o el·lipse, dins del qual hi haurà el nom d'aquest que descriurà la funcionalitat que representa. Un exemple de cas d'ús podria ser el següent:

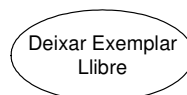


Figura B.2 : Representació d'un cas d'ús

- **Comunicacions o relacions entre actors i casos d'ús:** Cal que existeixi, com en qualsevol diagrama UML, un element que ens permeti representar relacions ja sigui entre actors, entre actors i casos d'ús, o bé, entre casos d'ús. La connexió entre aquests elements s'anomenarà comunicació. En la documentació s'ha utilitzat la manera més estàndard de relació a través d'una fletxa d'unió. Aquesta simbolitza el pas de missatges, o en alguns casos, representa que un determinat cas d'ús és l'especialització d'un altre. En aquests casos es representa a través d'una fletxa acompanyada de la paraula *extends*.

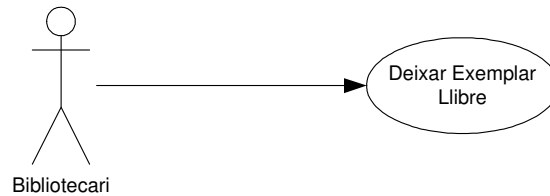


Figura B.3 : Comunicació entre actor i cas d'ús

- **Característiques addicionals:** Un diagrama de casos d'ús pot ésser ampliat a través de característiques addicionals que permeten recollir tota la informació relacionada amb el sistema (fronteres, especialització, generalització d'actors, generalització de casos d'ús, incusions i extensions) de manera eficient.

Per tal de simbolitzar la frontera entre el sistema i l'actor s'utilitza un rectangle. D'aquesta manera queden clarament delimitades les accions dels actors i les accions de les funcionalitats del sistema.

Les possibles relacions que es poden establir entre els actors poden ser de dos tipus: especialització, o bé, generalització amb herència. Els actors descendents, a través dels casos d'ús, hereten els rols i les comunicacions de l'actor pare.

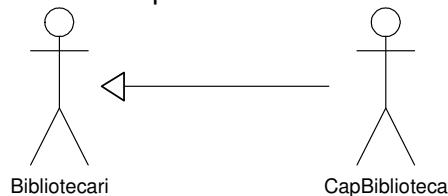


Figura B.4 : Relació d'herència entre actors

Entre els casos d'ús hi pot haver tres tipus de relació:

- **Generalització.** Una *generalització* d'un cas d'ús mostra que un cas d'ús *E* és un tipus especial d'un altre cas d'ús *G*. El cas d'ús *E* fa tots els processos del cas d'ús *G* més algun procés específic. Una generalització es simbolitza per mitjà d'una fletxa que apunta cap al cas d'ús *G* etiquetada amb <<generalize>>.
- **Inclusió.** Serveix per a mostrar funcionalitats comunes entre diversos casos d'ús i fer que el sistema utilitzi components preexistents. Aquesta relació s'anomena *d'inclusió*. El nou cas d'ús *I* és activat pels casos d'ús que l'inclouen. Per tant, no és un cas especial del cas d'ús original. Es fa servir una inclusió quan se sap exactament quan invocar un cas d'ús. En el diagrama es representa per una fletxa puntejada en direcció cap a *I*, amb una etiqueta <<include>>. El cas d'ús *I*, si és un cas d'ús complet, pot ser activat directament per un actor.
- **Extensió.** Un cas d'ús *E* es pot definir com una extensió opcional d'un cas d'ús base *B*: dins de *B* s'executa *E* quan es compleix una

condició determinada. Aquesta relació s'anomena *d'extensió*. I pot haver diverses extensions d'un mateix cas d'ús. En el diagrama es representa per una fletxa puntejada en direcció cap el cas d'ús *B*, amb una etiqueta `<<extend>>`. Dins del cas *B* s'hi poden posar *punts d'extensió* que serveixen per a determinar quan la utilització del cas *E* és apropiada.

Els diagrames de casos d'ús ens aporten una sèrie d'avantatges on possiblement uns dels més importants són:

- Proporcionen un llenguatge de comunicació entre usuaris i desenvolupadors.
- Faciliten la determinació de requeriments, així com la comprensió detallada de les funcionalitats del sistema.
- Ajuden a la generació de documentació d'usuari.
- Permeten la generació de casos de prova per a diversos escenaris.

A continuació es mostra un exemple senzill de cas d'ús per tal d'observar la comunicació entre els diferents elements que hi apareixen.

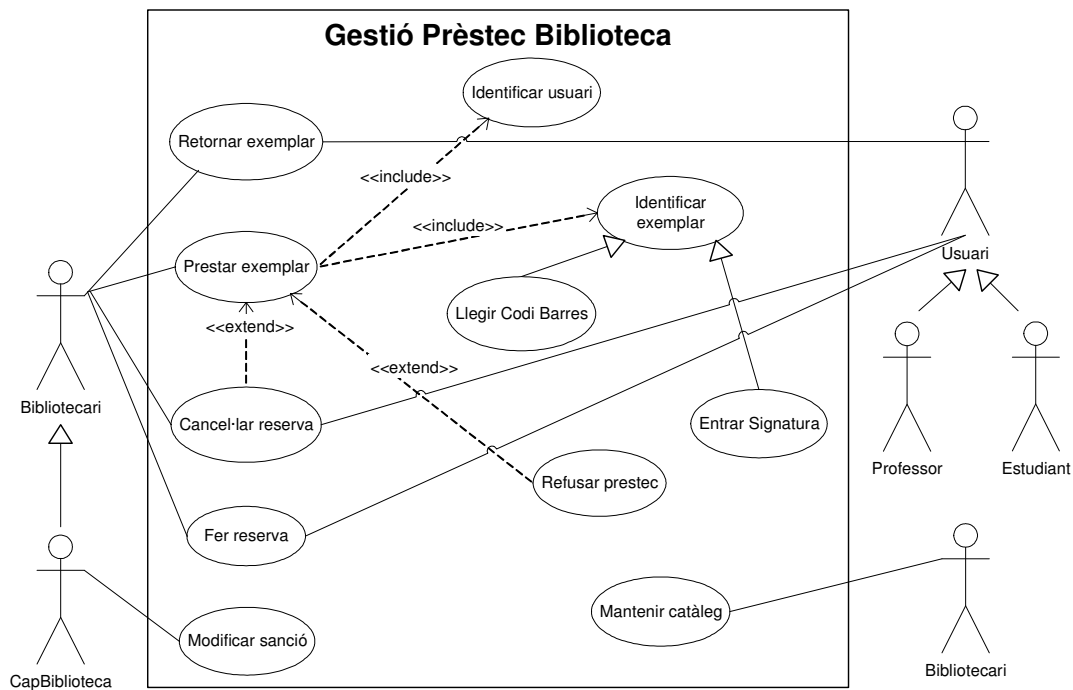


Figura B.5 : Exemple d'un diagrama de casos d'ús

B.2 Nomenclatura utilitzada en els diagrames d'activitat

Els **diagrames d'activitats** es centren en el flux d'activitats involucrades en un procés, generalment dins del marc d'un o diversos casos d'ús. Un diagrama d'activitats mostra en quin ordre s'executen les parts del procés i com depenen unes de les altres. Els diagrames d'activitat no proporcionen informació del comportament d'un objecte o de les col·laboracions entre objectes.

Tot seguit es descriuen els diferents elements que pot contenir un diagrama d'activitat:

- **Inici:** L'inici del diagrama d'activitat es representa a través d'un cercle negra situat a la part superior com el que es mostra a continuació.



Figura B.6 : Representació d'inici d'un diagrama d'activitat

- **Fi:** L'acabament s'indica a partir d'un cercle blanc/negre situat a la part inferior del diagrama.



Figura B.7 : Representació de fi d'un diagrama d'activitat

- **Activitats:** Les activitats que s'hi duen a terme durant el procés que es vol simbolitzar es representen mitjançant rectangles amb les puntes arrodonides. Tot seguit se'n pot veure un exemple:

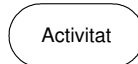


Figura B.8 : Representació d'activitat d'un diagrama d'activitat

- **Branques:** De cada activitat se'n deriva una transició que connecta amb la següent activitat i, a més a més, una transició pot derivar cap a diverses noves transicions (branques) mútuament excloents. Les expressions que controlen quina ha de ser la nova transició es posen dins de []. L'inici i el final de branca s'indica amb un rombe com el següent:

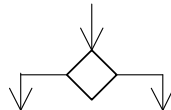


Figura B.9 : Representació de branques d'un diagrama d'activitat

- **Bifurcacions:** Una transició pot derivar cap a diverses activitats que es desenvolupen en paral·lel. La forquilla d'inici i la unió de retrobament es simbolitzen amb barres sòlides com les següents:

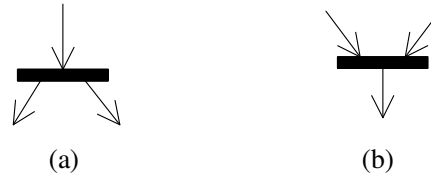


Figura B.10 : (a) Forquilla d'inici o inici d'activitats en paral·lel. (b) Forquilla de fi o unió de retrobament de les activitats

- **Carrers:** Els diagrames d'activitat es poden subdividir en carrers (*swimlanes*) per a mostrar el responsable (actor, objecte, unitat organitzacional, cas d'ús, ...) encarregat de l'activitat. Un carrer es simbolitza mitjançant un requadre o una línia de separació que representa una subdivisió del diagrama.

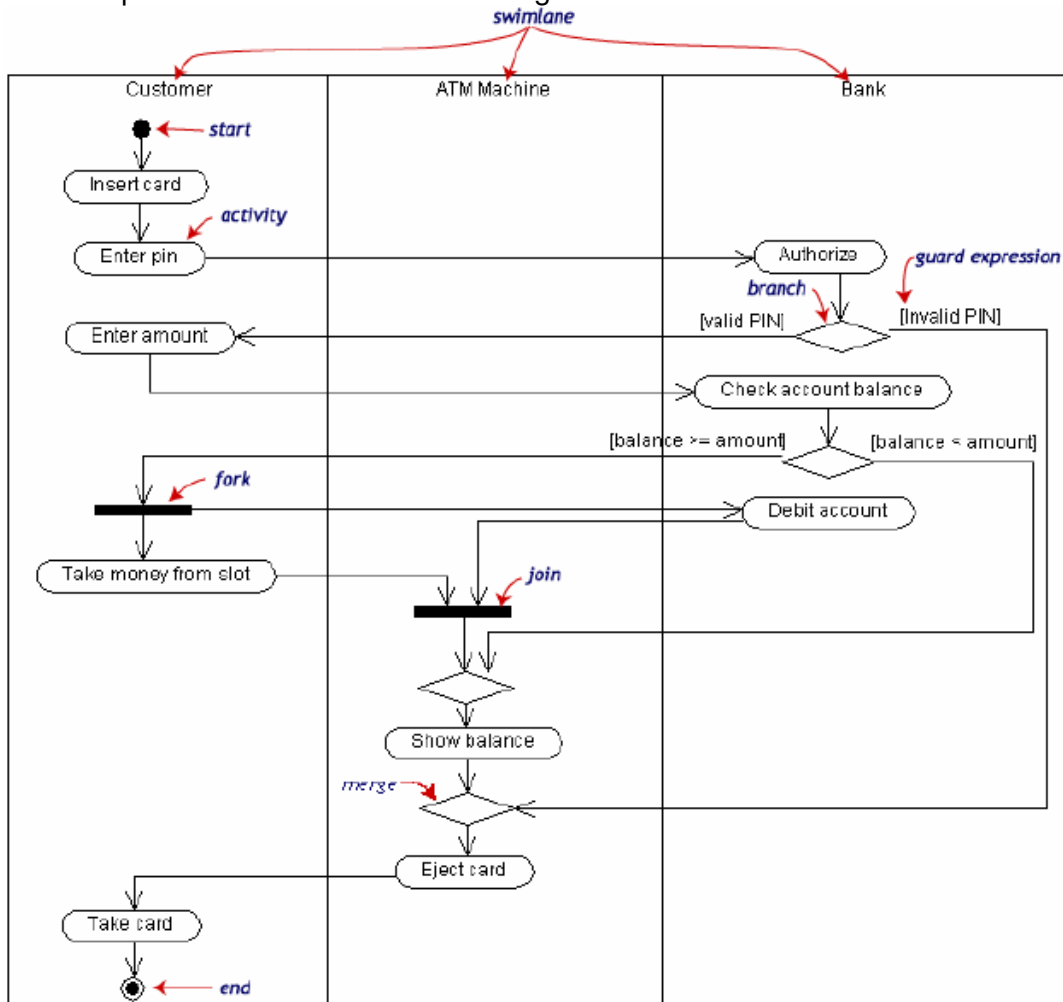


Figura B.11 : Exemple de diagrama d'activitat

Tots aquests elements descrits anteriorment s'uneixen entre si a partir de fletxes unidireccionals, indicant la direcció cap on viatja el flux d'informació.

B.3 Nomenclatura utilitzada en els diagrames de classes

Els **diagrames de classes** ens permeten explorar conceptes del domini del problema a la fase de definició de requeriments, analitzar/especificar requeriments a l'hora de construir el model d'anàlisi i descriure detalladament el programari a construir a la fase de disseny. Aquests ens proporcionen una visió estàtica del sistema a desenvolupar, ja que mostra les classes que interactuen sense informar sobre el que passa quan ho fan. Tot seguit es descriuen els diversos components d'un diagrama de classes.

- **Classes:** En UML una classe es representa per mitjà d'un rectangle dividit en tres parts horitzontals que contenen:

- la part superior: el nom de la classe
- la part central: els atributs de la classe. La sintaxi és la següent:

$$\underbrace{\text{visibilitat nomAtribut multiplicat: tipus = valorInicial}}_{\text{Única part obligatòria}}$$
- la part inferior: les operacions o mètodes de la classe. La sintaxi és la següent:

$$\underbrace{\text{visibilitat nomOperació}(\underbrace{\text{nomParàmetre: tipusParàmetre, ...}}_{\text{Llista de paràmetres}}): \text{tipusRetorn}}_{\text{Signatura de l'operació}}$$

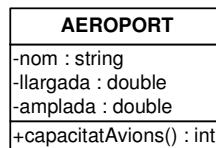


Figura B.12 : Representació d'una classe

En l'exemple anterior podem veure l'esquema que segueix una determinada classe:

- S'anomena Aeroport
- Disposa dels atributs llargada i amplada que són reals (*double*) i de l'atribut nom que és una cadena de caràcters o *String*.
- Finalment, com a mètodes n'hi ha un anomenat *capacitatAvions()* que retorna el número d'avions que caben dins de l'àrea que ocupa l'aeroport.

- **Classes abstractes:** Existeix una petita distinció a l'hora de representar de forma esquemàtica les classes abstractes respecte de les que no ho són. Aquesta diferència es troba tan sols en què el nom de la classe apareixerà en cursiva. Per tal d'indicar que un mètode és abstracte davant del tipus de retorn posarem la paraula clau *virtual*.

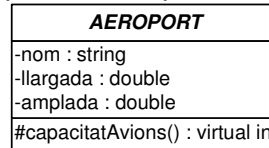


Figura B.13 : Representació d'una classe abstracta

- **Visibilitat i accés als mètodes i atributs:** La visibilitat d'un atribut o d'una operació indica com aquests poden ser accedits des d'altres classes. Aquesta s'especifica mitjançant el símbol que tenen tant els atributs com els mètodes, abans del nom (-, +, #, ...):
 - **Símbol + :** En aquest cas l'accés al mètode o atribut que l'acompanyi serà *public*. Això significa que serà totalment visible, és a dir, que podrà accedir-hi qualsevol objecte de l'aplicació.
 - **Símbol - :** Si un atribut o una operació va precedida per un signe negatiu significa que és privat/da i, per tant, tan sols hi podem accedir-hi des de la pròpia classe. Fora d'aquesta és com si no existís l'atribut.
 - **Símbol # :** Aquest símbol fa referència a un atribut o un mètode protegit (*protected*). Només podem accedir-hi des de la mateixa classe, o bé, qualsevol de les seves subclasses (filles) que heretin d'ella. Per aquest motiu utilitzar atributs o operacions amb aquest tipus de visibilitat té sentit en classes abstractes.

En el cas que la visibilitat no s'especifiqui explícitament, per defecte els atributs seran privats i els mètodes públics.

- **Package (Paquets):** Un paquet és una col·lecció d'elements del model (classes, altres paquets, casos d'ús, ...) relacionats lògicament. Es tracta d'un element de UML que serveix per agrupar una sèrie de classes que disposen d'aspectes en comú.

Els paquets es representen per rectangles amb petites pestanyes a la part superior i el nom d'aquest es posa dins del rectangle o pestanya.

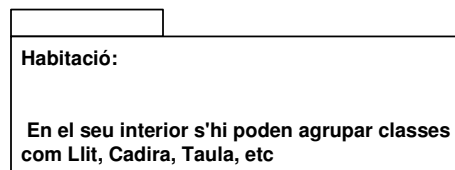


Figura B.14 : Representació d'un paquet

Tal i com es pot comprovar amb la *figura B.14*, el paquet habitació permet agrupar totes aquelles classes que estiguin relacionades com podrien ser Llit, Cadira o Taula.

La utilització de paquets ens permet fer-nos una idea de la distribució de l'aplicació tot i que no té perquè tenir una gran incidència sobre la implementació final.

- **Relacions entre classes:** Per tal de construir un diagrama de classes calen reflexar-hi les relacions existents entre les diferents classes que hi intervinguin. S'utilitzaran diversos tipus de fletxes cadascuna amb el seu corresponent significat. Les relacions més importants utilitzades en la documentació són les següents:

- Generalització / Especialització: Es tracta d'una de les relacions més utilitzades en el paradigma de programació orientat a objectes (OO). Ens permet representar una relació d'herència en la qual una classe fa la funció de superclasse (mare) davant d'altres subclasses (filles). En aquests casos les classes filles hereten (pot utilitzar com si fossin seus) tots els atributs i mètodes de la classe mare (sempre que siguin públics o protegits).

Una generalització/especialització es representa per un segment amb un extrem triangular (fletxa buida) apuntant cap a la superclasse. En l'exemple que es mostra a continuació es pot comprovar com les classes Estudiant, Professor i Alumne són subclasses de la superclasse Persona. El mètode que conté la classe mare anomenat *assignarDNI(...)*, és accessible des de qualsevol de les classes filles ja que és públic. Per tant, podrà ser utilitzat com si fos propi.

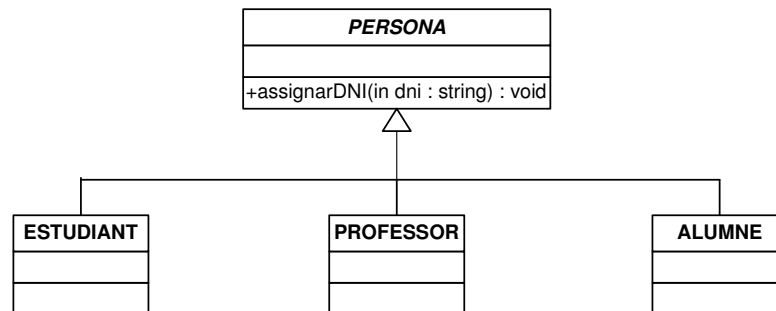


Figura B.15 : Relació Generalització/Especialització

En el diagrama no cal especificar quins mètodes i atributs s'hereten ja que se sobreentén que tots els de la classe mare són heretats per les classes filles.

- Associació: Interrelació entre instàncies de dues classes. Una instància d'una classe ha de disposar d'informació d'una instància de l'altre classe per a poder fer el seu treball. En un diagrama, una associació es representa a través d'un segment que connecta

les dues classes. En un dels dos extrems d'una associació s'hi pot posar el nom d'un rol per tal de clarificar la natura de l'associació. Per tal de mostrar el sentit de navegació en el qual es poden realitzar consultes s'afegeixen fletxes, mostrant així qui és el propietari de l'associació. Aquelles associacions que no tenen sentit seran bidireccionals. Existeixen diversos tipus d'associació, com són:

- **Composició:** L'associació per composició és un tipus de relació estàtica. Es tracta d'una associació en la qual el temps de vida de l'objecte inclòs està condicionat pel temps de vida de l'objecte que l'inclou. És a dir, l'objecte base es construeix a partir de l'objecte inclòs. Una composició es representa per un segment amb un extrem en forma de diamant ple apuntant cap a la classe feble.
- **Agregació:** Es tracta d'una associació en la qual es distingeixen un tot i una part: una instància d'una classe (tot) es relaciona amb una col·lecció d'instàncies de l'altre classe (part). Una agregació es representa per un segment amb un extrem en forma de diamant buit apuntant cap a la classe singular.

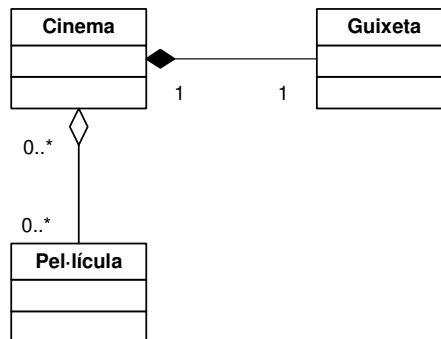


Figura B.16 : Associació per composició (Cinema – Pel·lícula) i per agregació (Cinema – Guixeta)

- **Relació de dependència:** Una dependència és una relació entre dues classes en la qual un canvi a una de les classes pot forçar canvis a l'altra classe. En els diagrames de classes les dependències s'indiquen amb una fletxa puntejada. Per exemple, en el següent cas la classe Companyia depèn de la classe Cooperativa (la cooperativa utilitza la classe companyia):

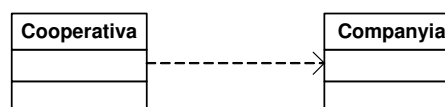
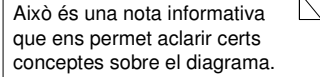


Figura B.17 : Relació de dependència

- **Notes:** En alguns casos interessa aclarir determinats conceptes, o bé, afegir informació addicional per tal de facilitar la comprensió del diagrama. Per aquest motiu, qualsevol diagrama UML pot anar acompanyat d'alguna nota aclaridora en forma de text dins d'un rectangle amb un extrem doblat com la següent:



Això és una nota informativa
que ens permet aclarir certs
conceptes sobre el diagrama.

Figura B.18 : Representació d'una nota

B.4 Nomenclatura utilitzada en els diagrames de seqüència

Un **diagrama de seqüència** és un diagrama d'interacció que detalla com s'executen les operacions en funció del temps: quins missatges són enviats, per quin objecte, a quin objecte i quan.

- **Objectes:** Els objectes dins dels diagrames de seqüència són els elements bàsics i representen instàncies de les classes. Aquests es simbolitzen a través d'un rectangle dins del qual hi consta el nom de la classe al qual fa referència subratllat (en alguns casos es pot indicar el nom_objecte:nom_classe). Per exemple, en la següent figura es mostra un objecte que pertany a la classe Objecte:



Objecte

Figura B.19 : Representació d'un objecte d'un diagrama de seqüència

- **Línia de vida:** Un dels aspectes més importants que ens permeten representar els diagrames de seqüència és el temps o l'ordre en el qual es duen a terme les accions. Per aquest motiu existeix una línia vertical puntejada anomenada línia de vida que representa el temps durant el qual un objecte existeix.
- **Barra d'activació:** Aquest element representa el temps d'execució del missatge i es simbolitza a través d'un rectangle vertical situat al llarg de la línia de vida.
- **Tipus de classes d'anàlisi:** Existeixen tres tipus diferents de classes d'anàlisi en UML que són les següents:
 - Classes d'entitats: Contenen els objectes bàsics que corresponen al món real. Generalment els objectes d'aquestes classes no necessiten conèixer res uns dels altres. Representen, per tant, objectes que poden trobar-se físicament, com per exemple: Persona, Taula, Llit, ...



Classe d'entitat

Figura B.20 : Representació d'una classe d'entitat

Restriccions: Les classes d'entitat tan sols poden interactuar amb les classes de control i les classes de frontera.

- Classes de control: Cada cas d'ús ha de tenir com a mínim una classe de control que s'ocupi de la comunicació entre objectes. L'objecte de control serà l'encarregat de dirigir els diferents camins del cas d'ús.

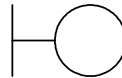


Classe de control

Figura B.21 : Representació d'una classe de control

Restriccions: Les classes de control poden interactuar amb les classes d'entitat, de frontera i de control.

- Classes de frontera: Defineixen les interfícies amb els actors, és a dir, finestres, diàlegs per pantalla i menús. Aquestes classes tenen la tasca de fer de pont entre els usuaris i l'aplicació.



Classe de frontera

Figura B.22 : Representació d'una classe de frontera

Restriccions: Les classes de frontera poden interactuar amb els actors, les classes de control i les classes d'entitat. Els actors només podran interactuar amb aquestes classes, les de frontera.

- **Relacions entre objectes:** En la creació dels diagrames de seqüència que es mostren en l'apartat d'anàlisi de la documentació hi apareixen tres tipus de relacions entre objectes:
 1. *Pas de missatges:* Representen crides a mètodes de l'objecte al qual van destinats. Es simbolitzen mitjançant fletxes estàndards sobre les quals s'especifica el mètode cridat. Aquesta anirà des de l'objecte que realitza la crida fins al que la rep.
 2. *Retorn:* En finalitzar l'execució d'una crida es retorna a l'objecte que l'ha realitzat a través d'una fletxa puntejada. Aquesta anirà cap a l'objecte origen que, en el seu moment, va realitzar la crida.
 3. *Iteració:* Es tracta d'una operació que es realitza sobre un mateix objecte i que és important per l'aplicació.

En el següent exemple es poden observar cadascuna de les relacions utilitzades en els diagrames realitzats:

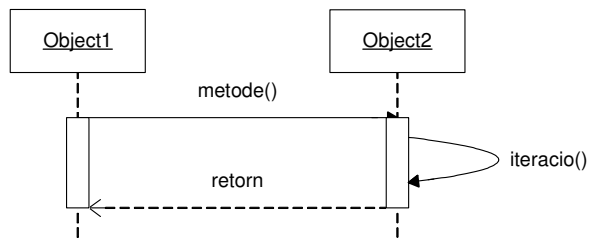


Figura B.23 : Exemple de diagrama de seqüència